SYSTEMS BANK

アスキー・システム・バンク（PC88＃1）

PC88＃1

PC-8801 BASIC入門

工藤丈彦・屋敷誠二・横溝和宏共著

アスキー出版局

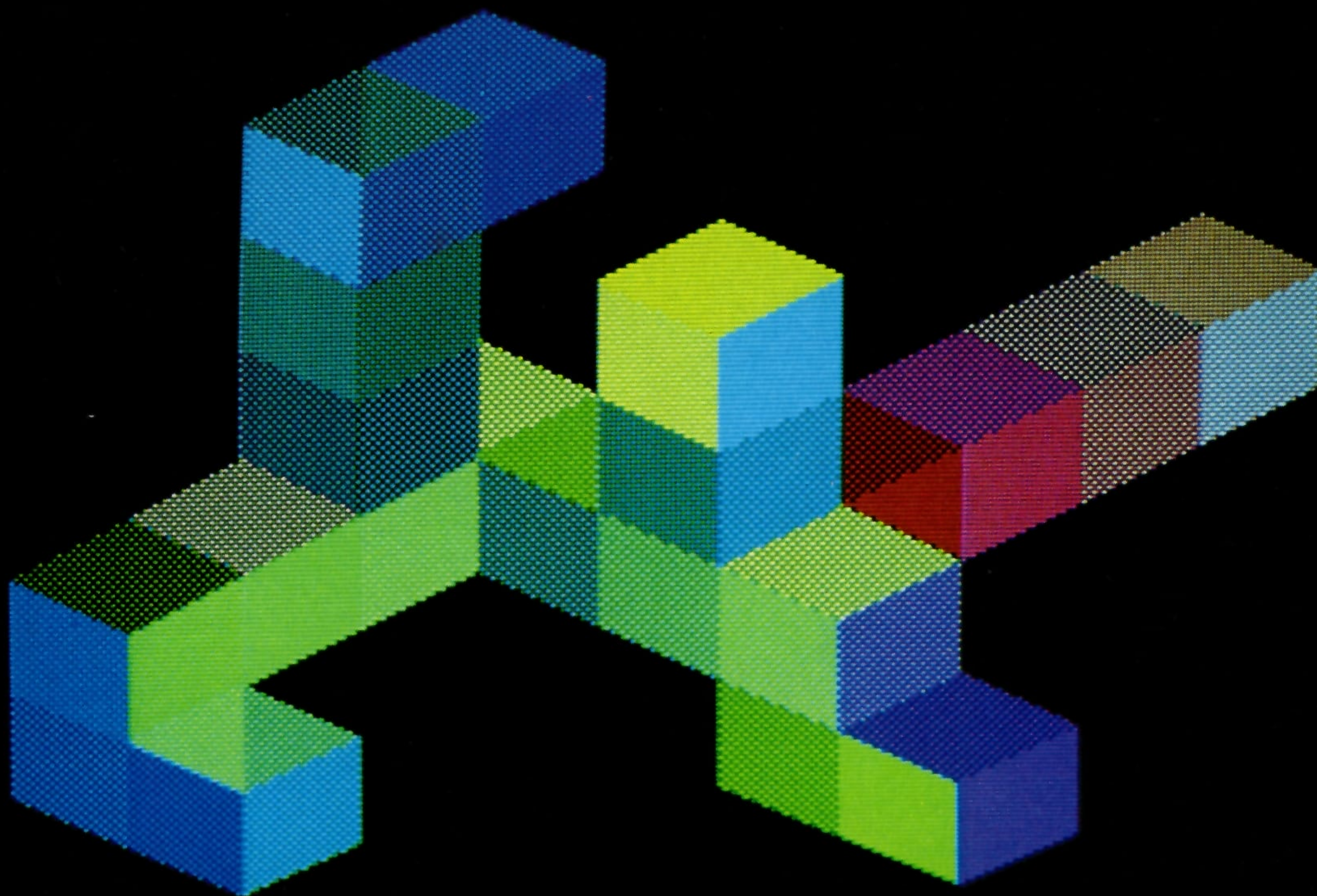
アスキー・システム・バンク

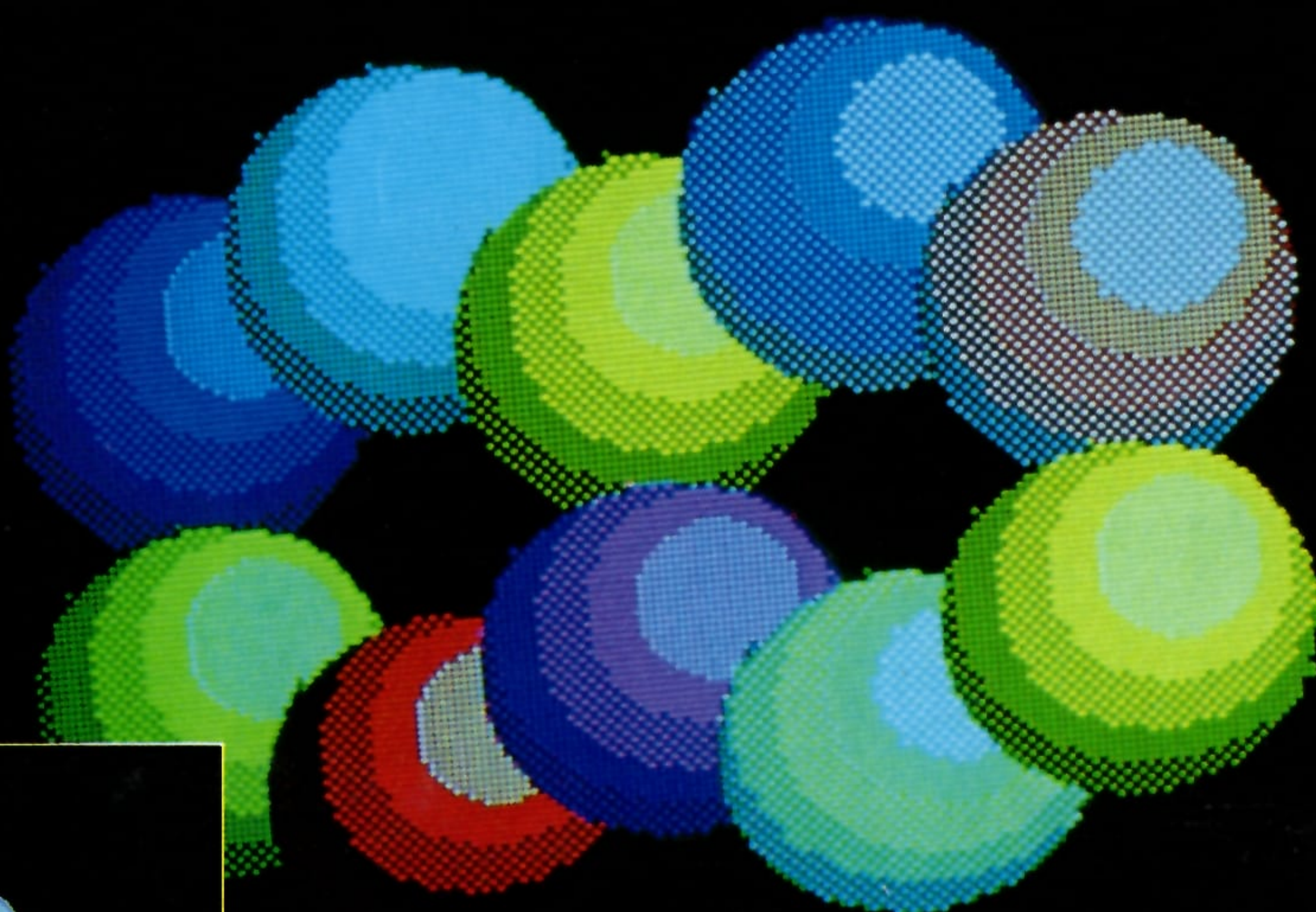
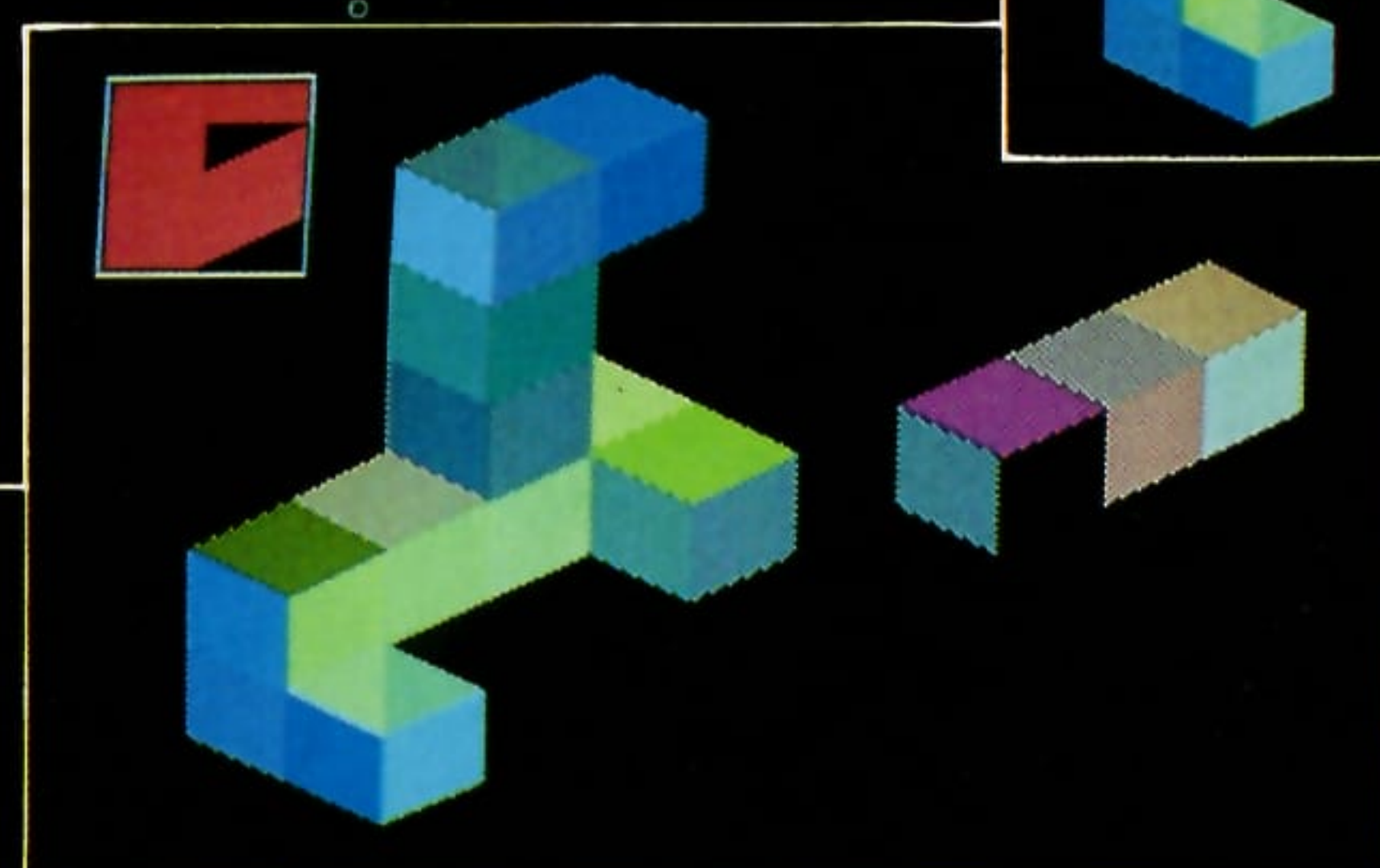
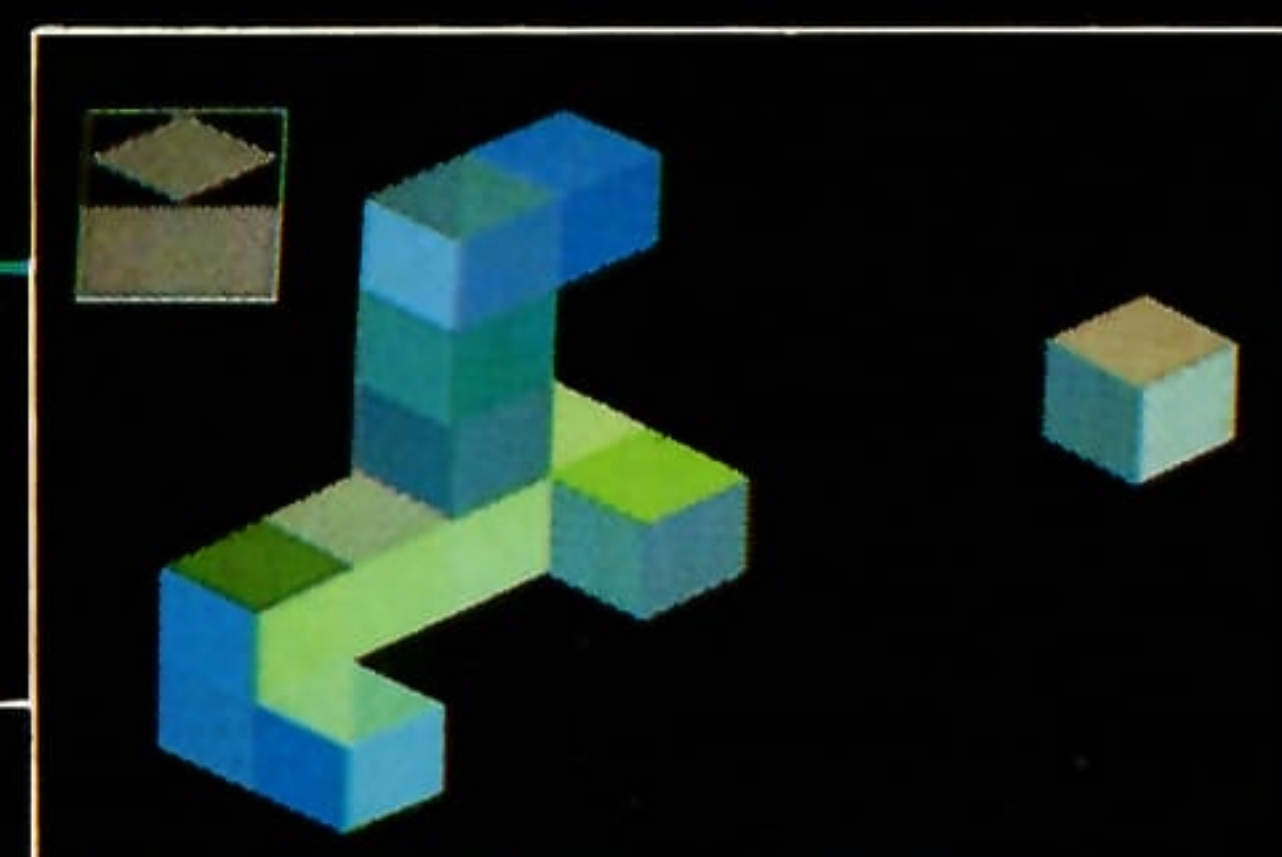
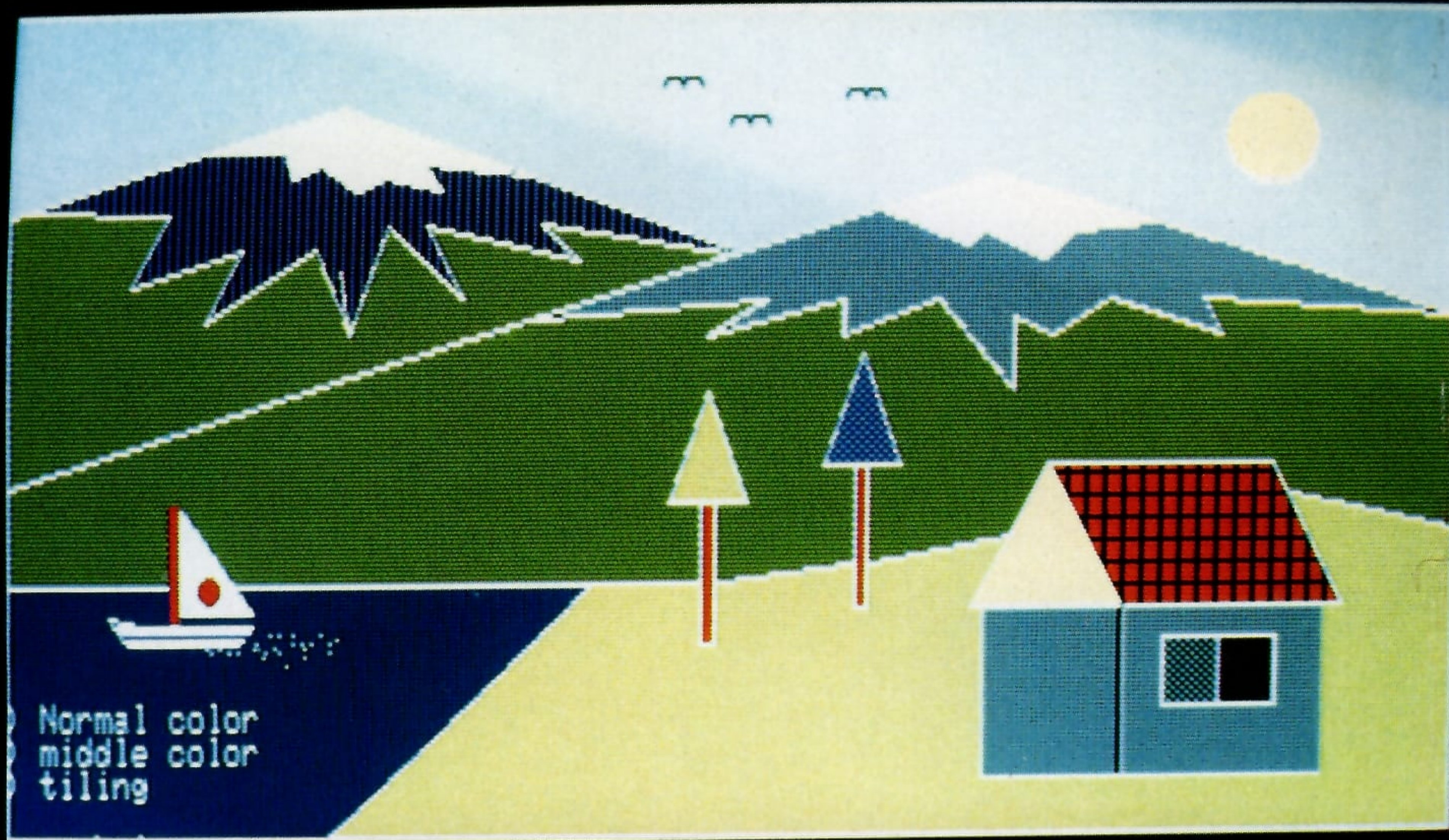
PC-8801 BASIC入門

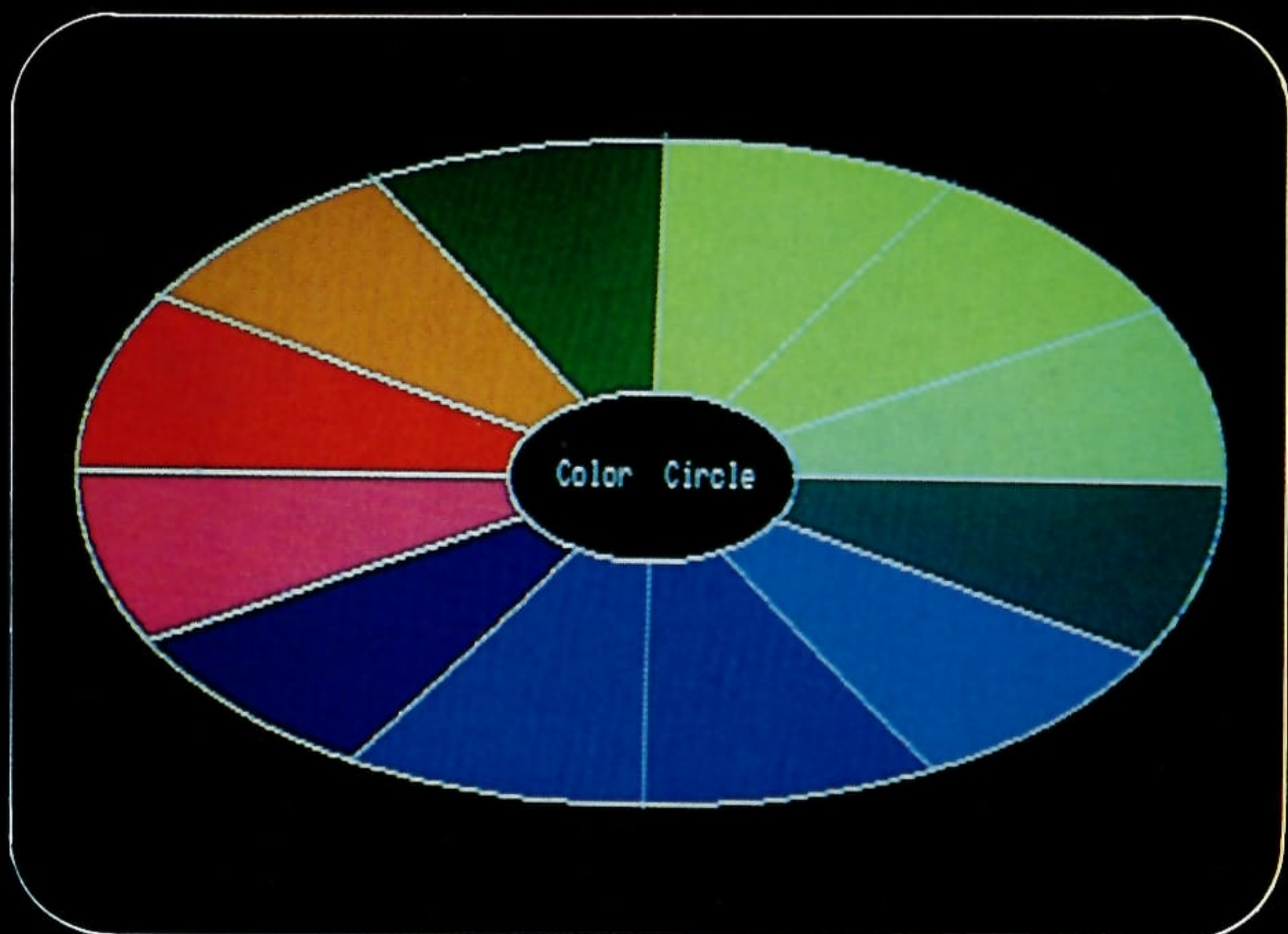
工藤丈彦・屋敷誠二・横溝和宏 共著

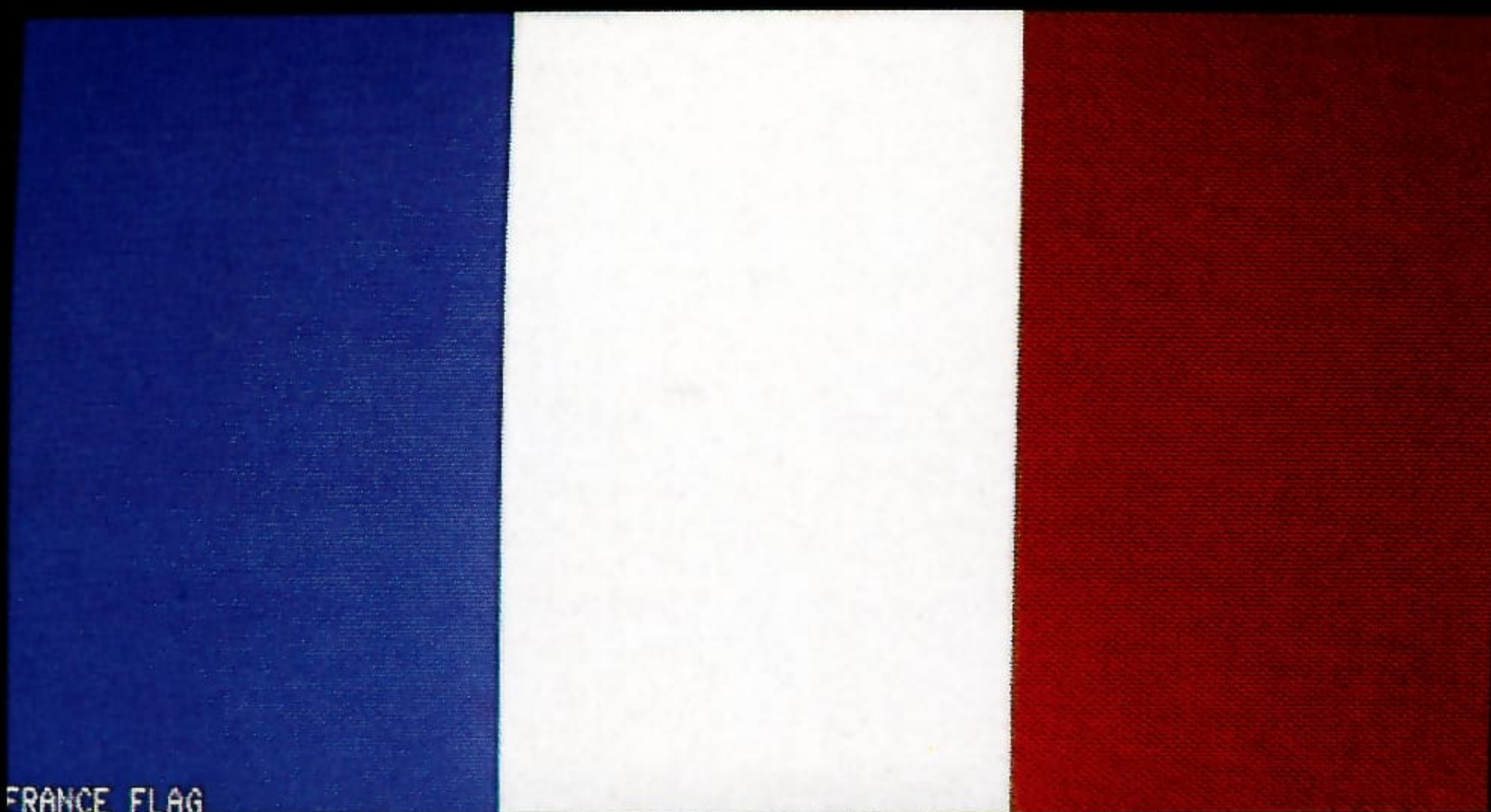
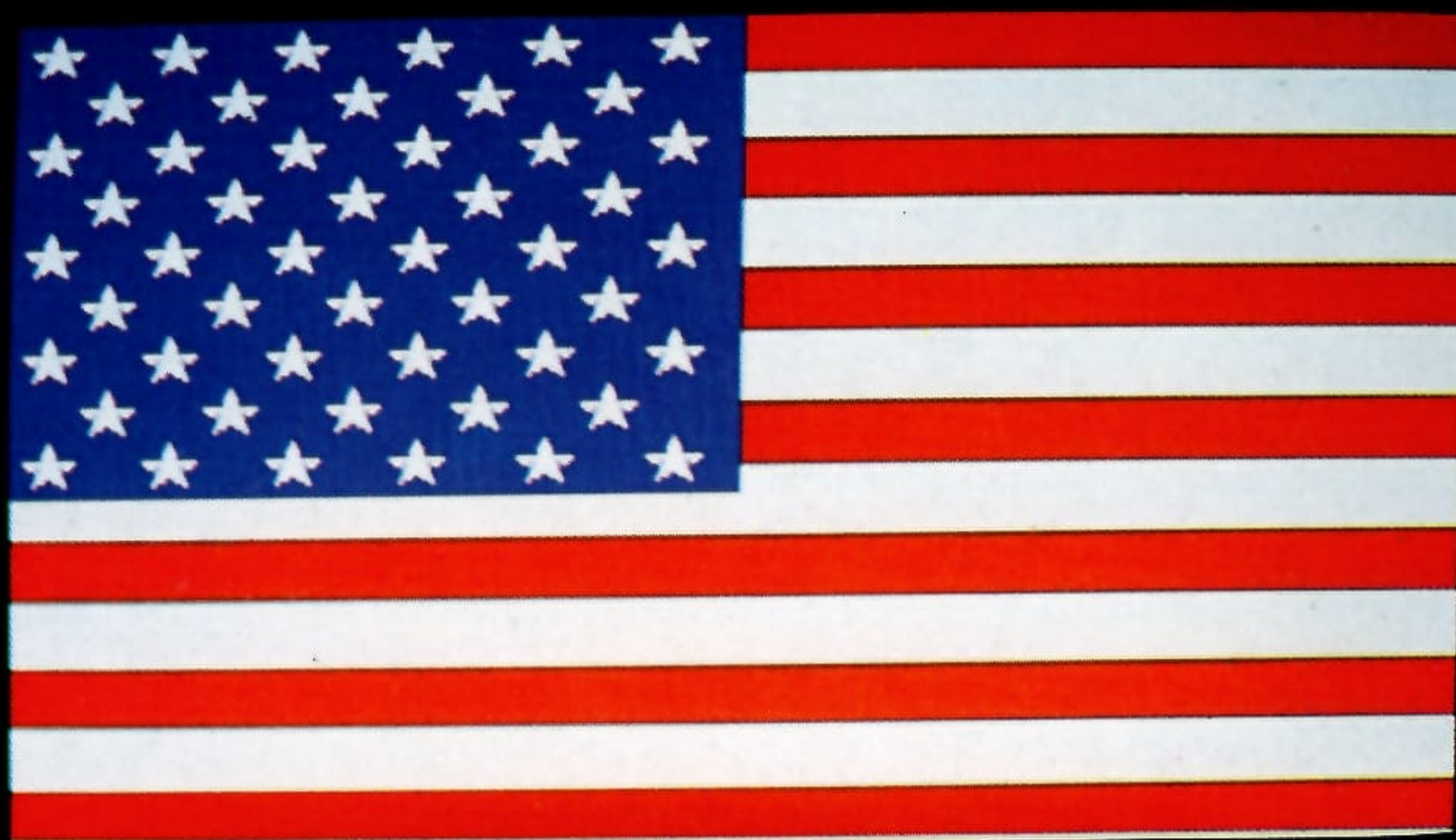
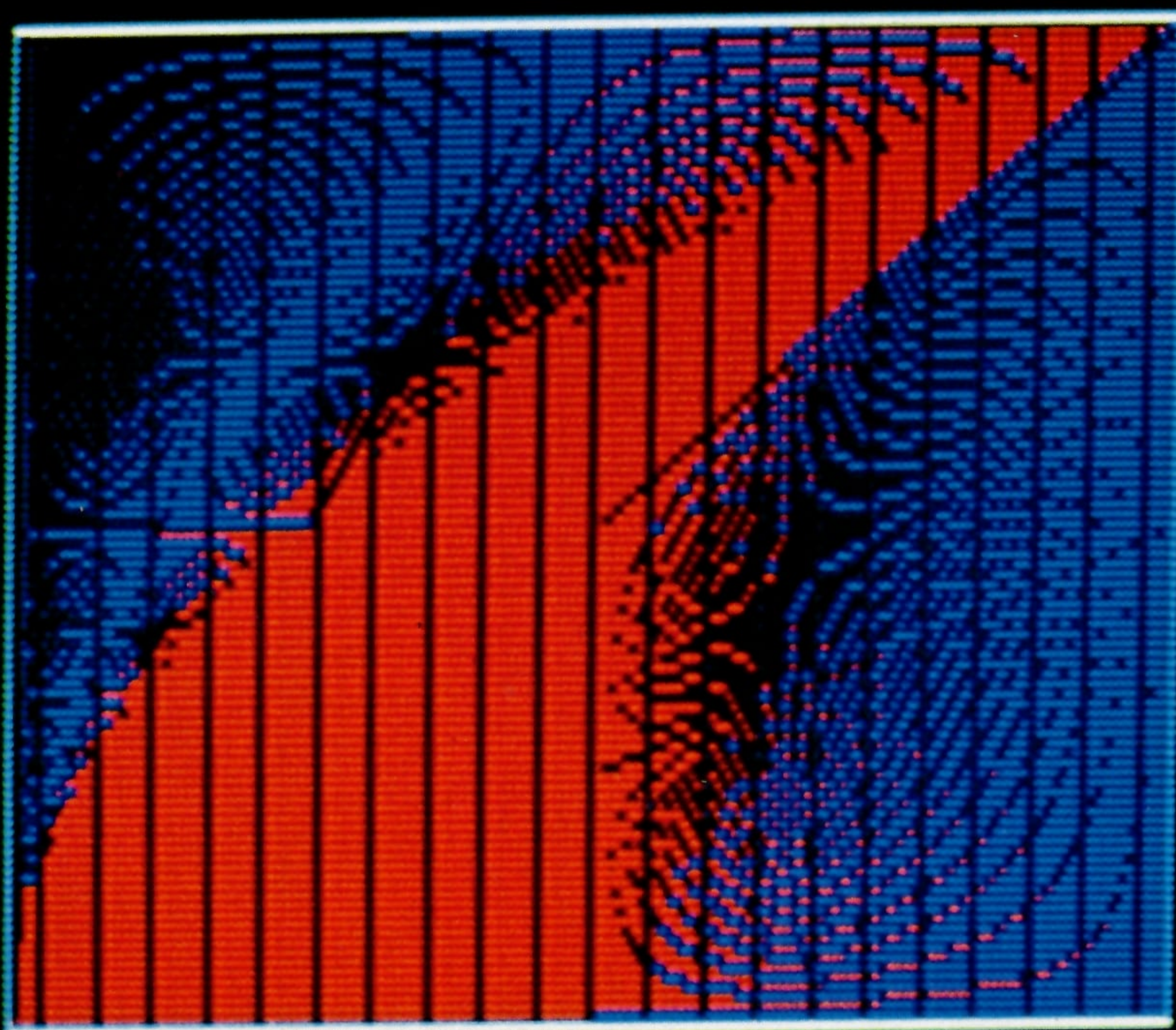
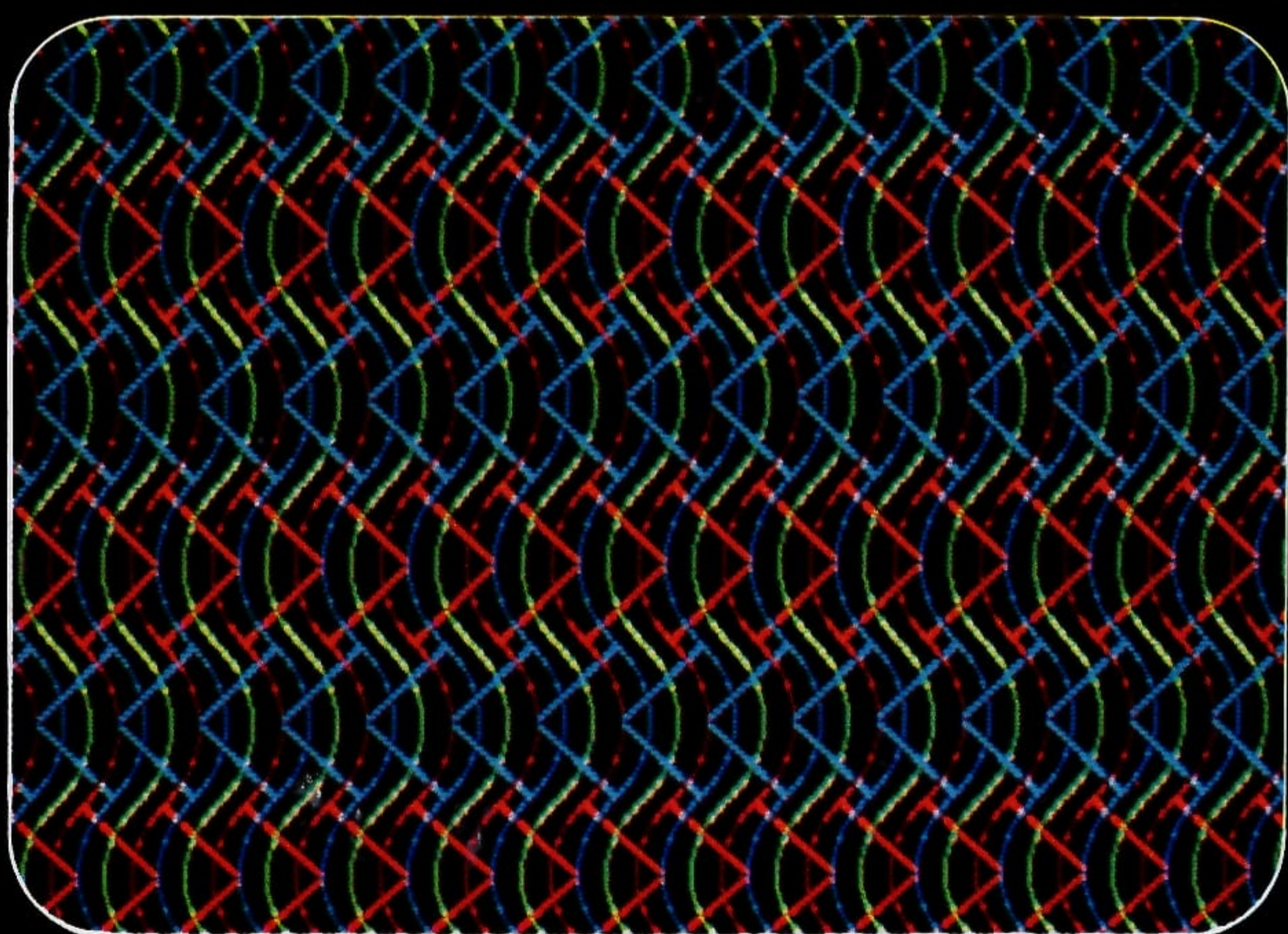
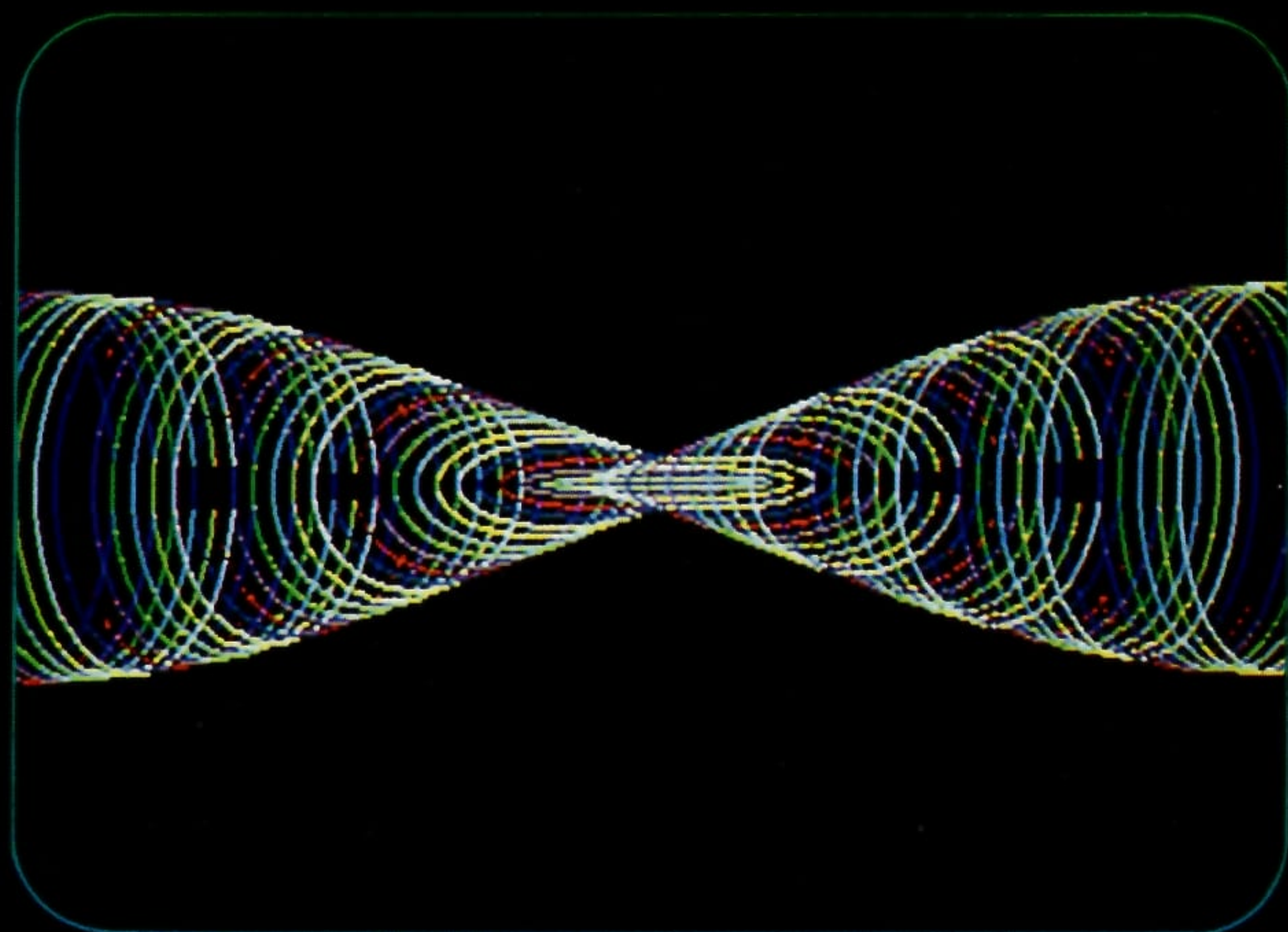
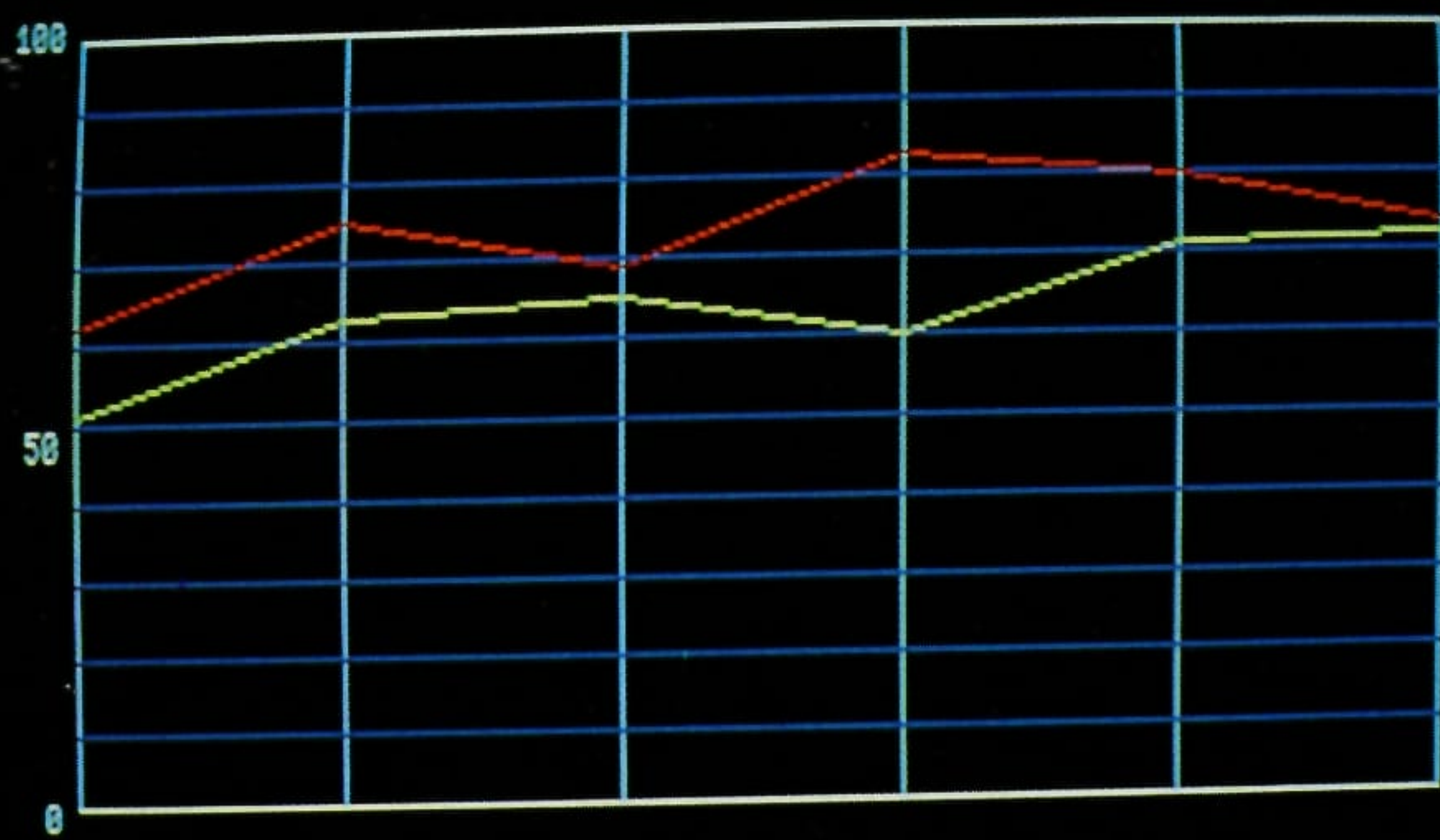
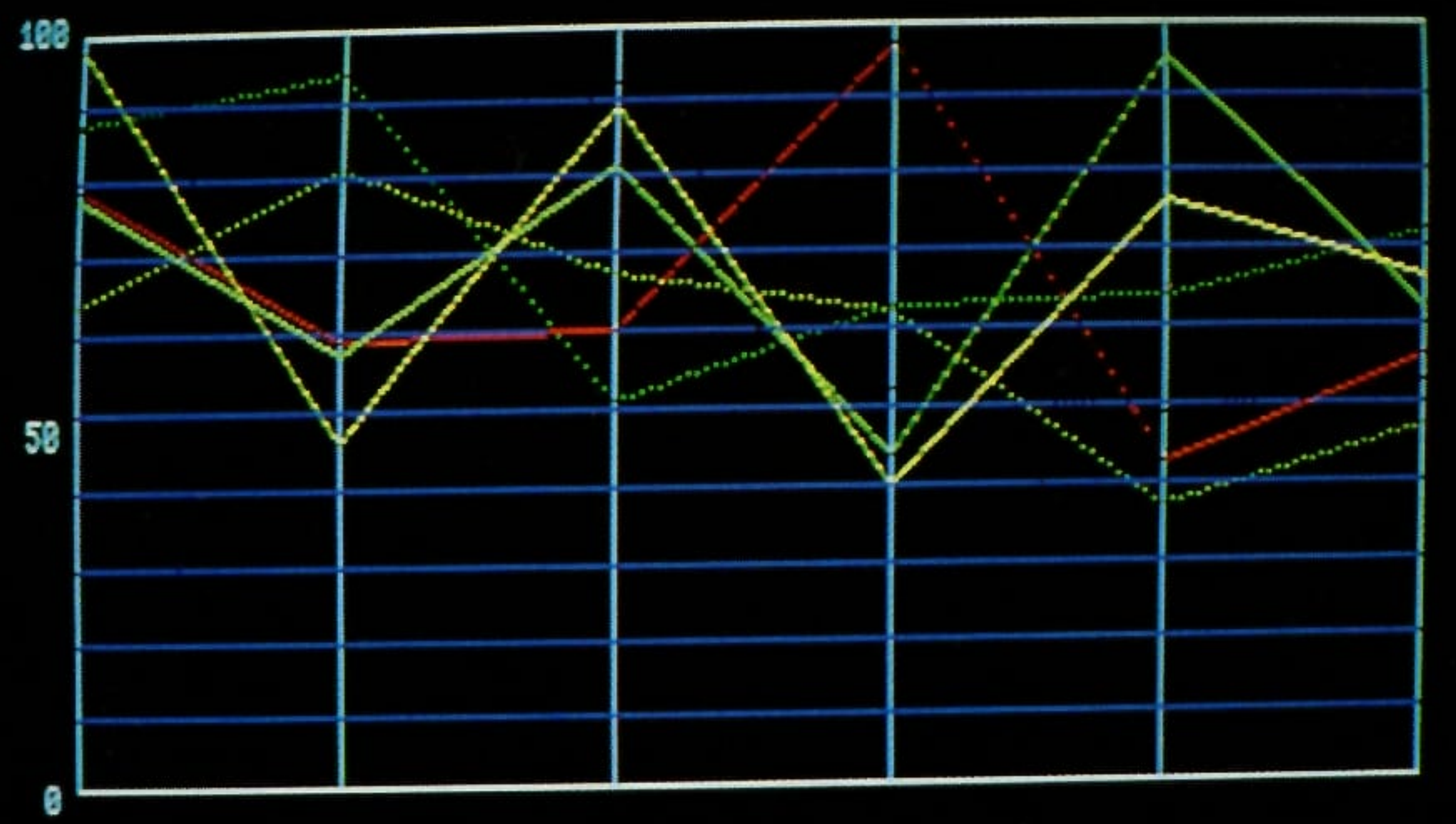
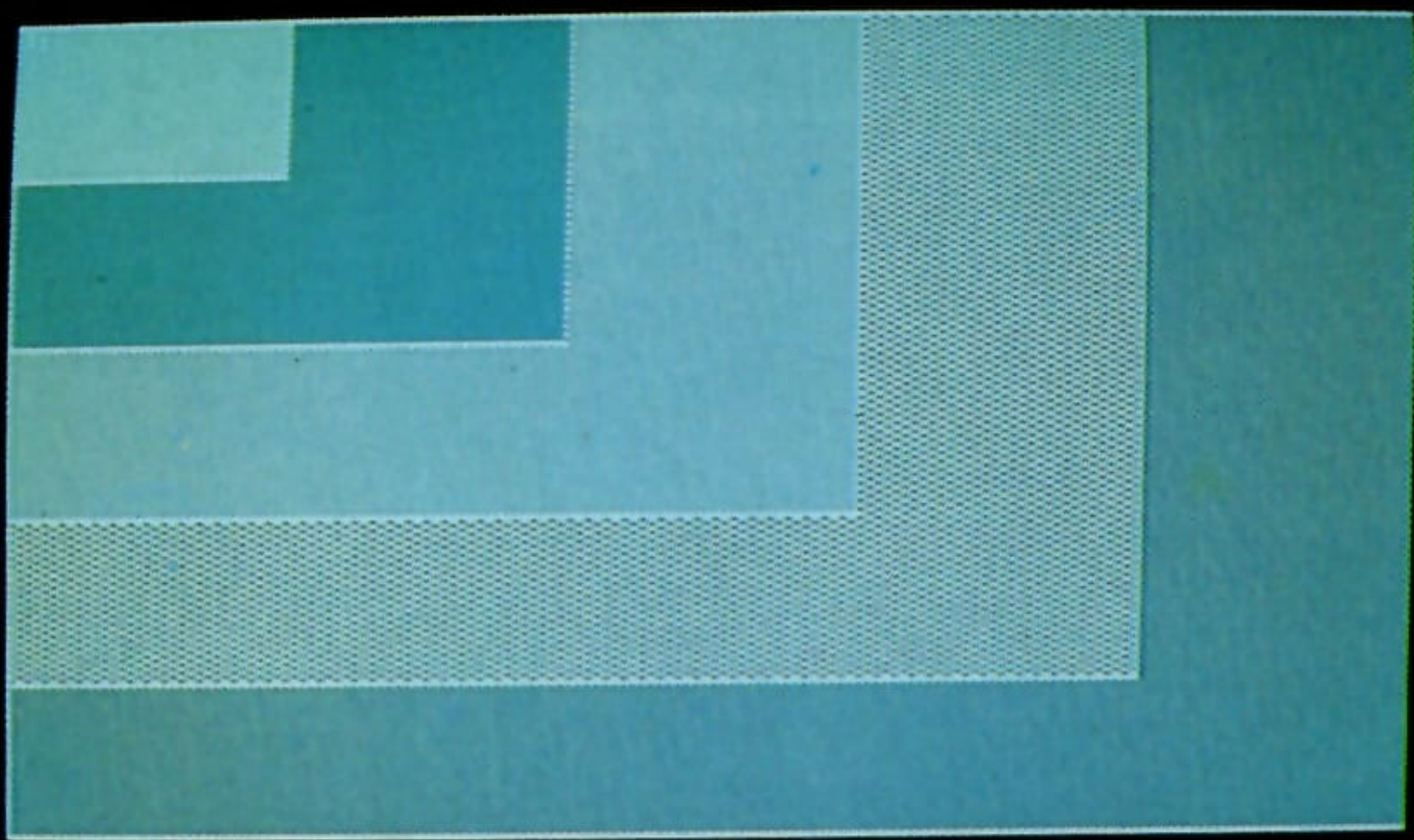
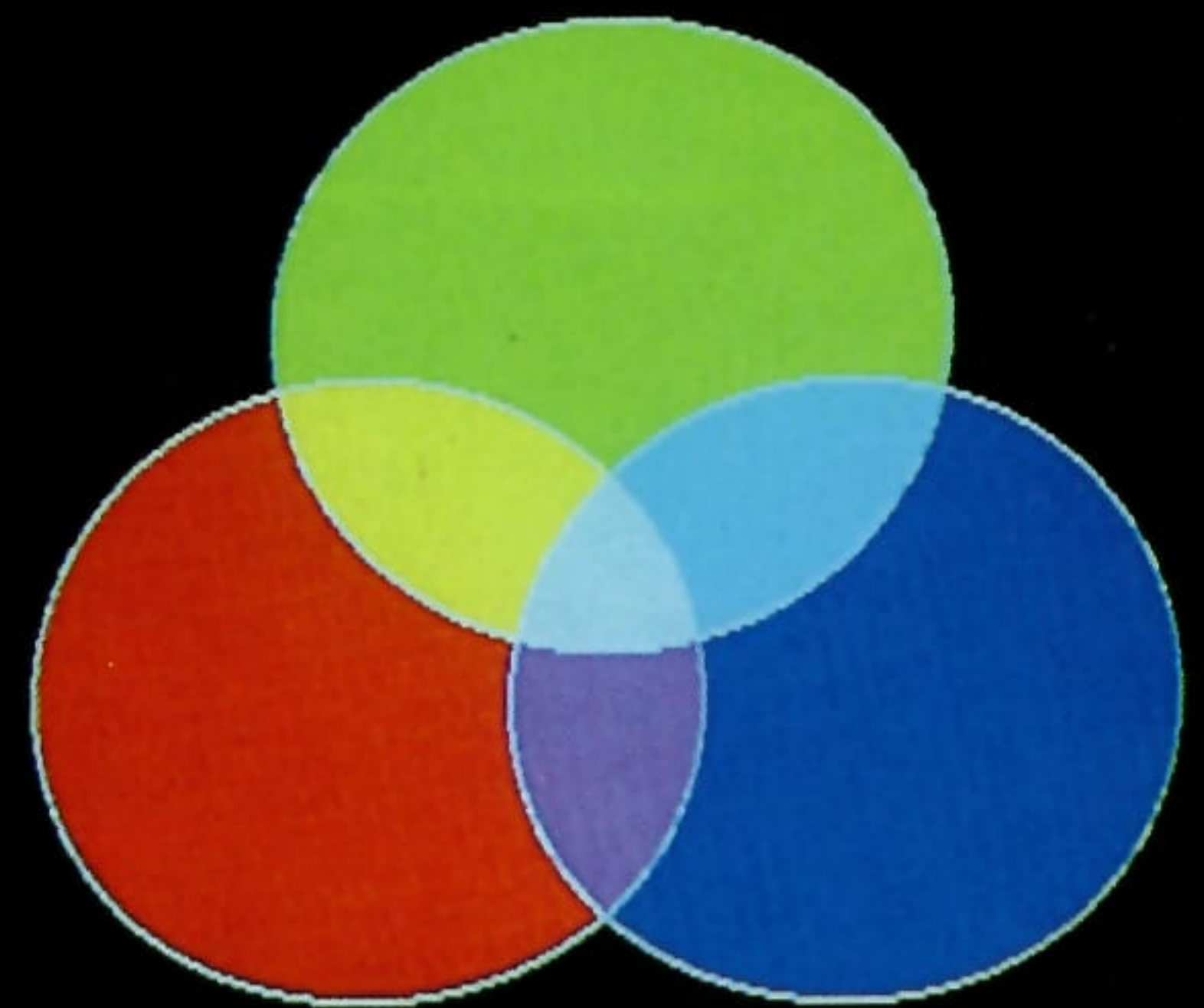
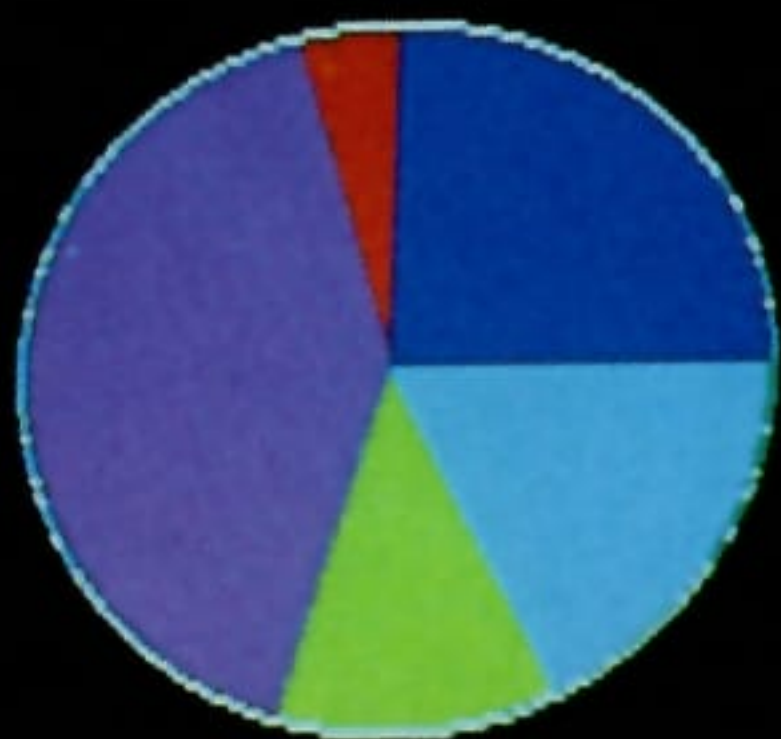
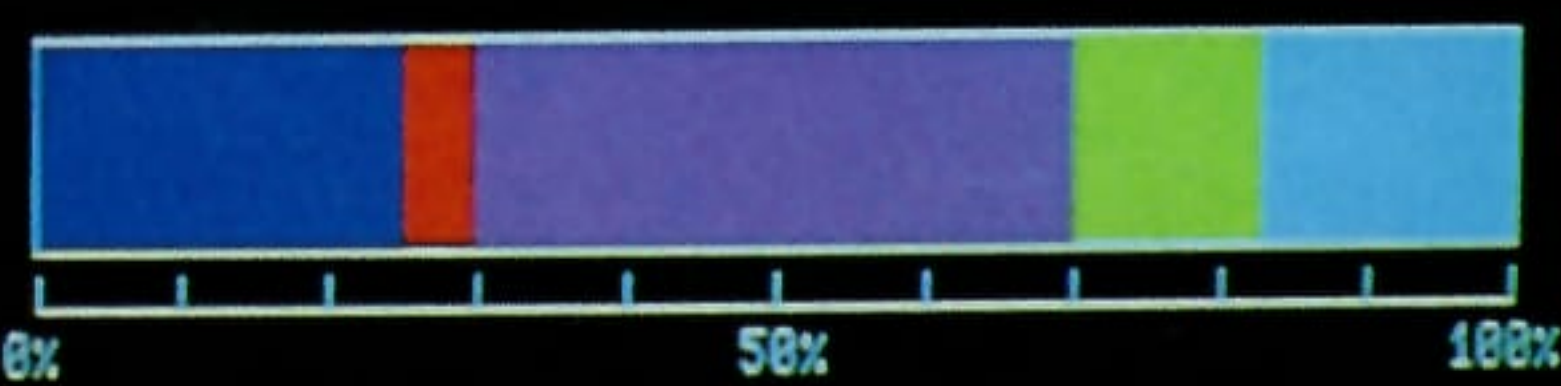
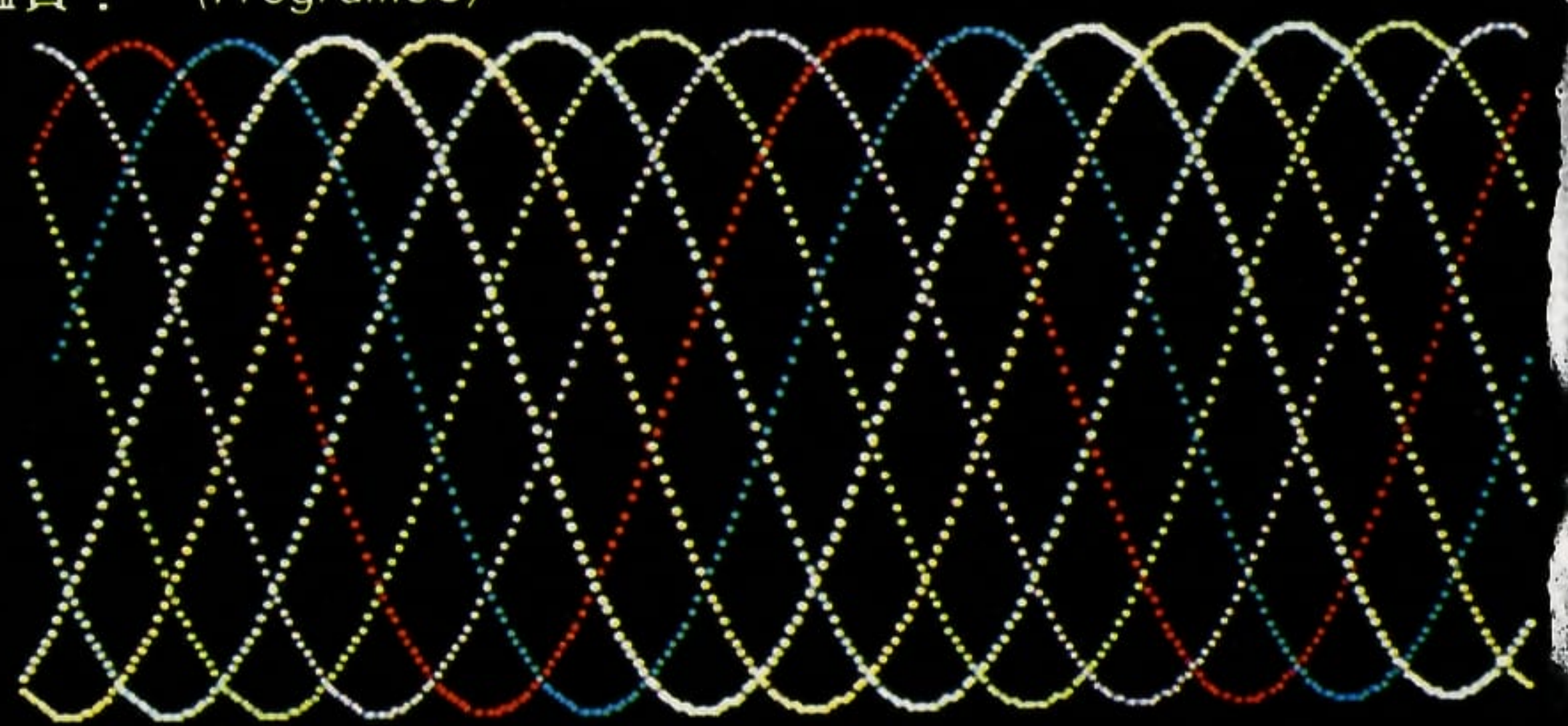
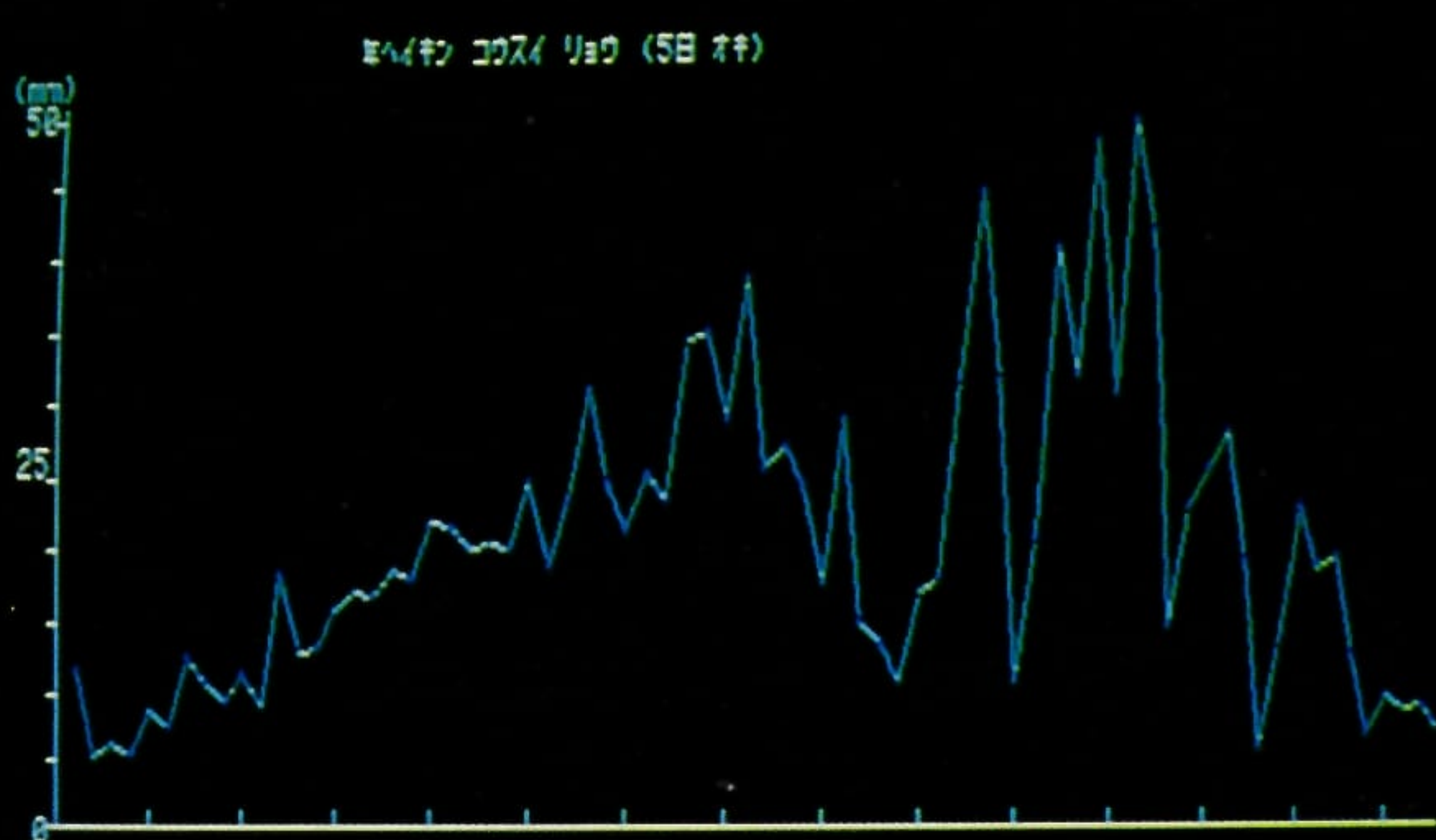
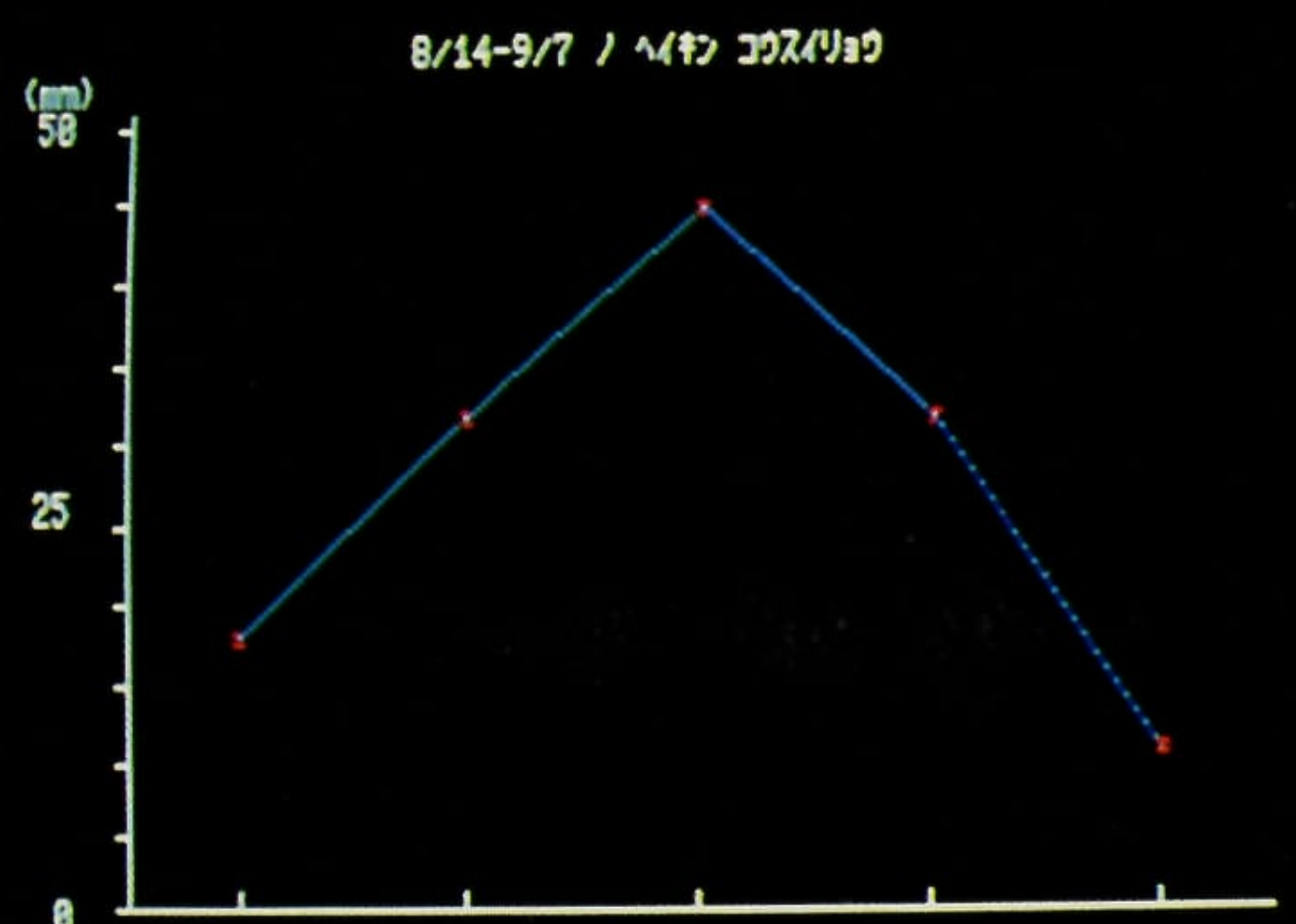
PC88＃1

ASCII SYSTEM'S BANK

Photo by Akio Kambayashi
Bill Gillette／Imperial Press（背景写真）

上）応用例１　フタのないboxが２つ見えます
下）応用例２　CPUの分子構造？

ここに載せた写真はすべてPC-8801上で実際に実行された結果を撮影したもので，あなたをグラフィックスの世界へ導く指針となることでしょう．

カラー・パレットを変えるだけで，このとおり　（このプログラムは巻末のAPPENDIXに掲載されています）

テレビ黒板を使って　(Program36)

これだけの色が簡単に使えます　(Program20)

カラーが微妙に違います　(Program19)

Random Seedは幾つでしょう　(Program27)

国旗いろいろ　*(Program21)*

星条旗もりっぱに描けます　*(Program34)*

時間がかかった分だけ楽しめます　*(Program33)*

CIRCLE文(扇形)を集めて　*(Program14)*

CIRCLE文(円)を集めて　*(Program15)*

PC-8801 BASIC入門

工藤丈彦　屋敷誠二
横溝和宏　　　共著

アスキー出版局

この本を読む前に

本書は，PC-8801の機能，及び，そのメインプログラミング言語であるN88-BASICについて，次の様な方々を対象として書かれています．

今までBASIC言語を知らなかったが，勉強してみようと思った方．

BASIC言語を使ったことがあり，ある程度は知っているが，このPC-8801について詳しく知り，活用したい方．

N88-BASICの持つ，グラフィック機能についてよく知りたい方．

N88-BASICは一通りマスターしたが，今１つ，プログラム作りのノウハウが掴めない方．

そして，すでにPC-8801を十分活用している方にとっては，座右の書となるよう書かれています．

本書を書くにあたって使用したシステムは，本体＋高解像度カラーモニタ＋ミニフロッピーディスク（片面用）であり，使用した言語は，N88-DISK BASICです．N88-DISK BASICは，ROM版のN88-BASICにくらべ幾つかの拡張された命令がありますが，基本的には同一です．したがって，どちらのユーザーの場合も十分に適用できます．ただし，N88-DISK BASIC固有の命令は，注意書きを添えてありますから，気を付けるようにしてください．

次に，本書の構成について一通り紹介しておきましょう．

１章　PC-8801のプロフィール

BASIC，プログラミングなどの具体的な話に入る前に，まずPC-8801というコンピュータがどんな構成のもとでどんな特色を持って動いているのかをハードウェアとソフトウェアの両視点から光を当てて紹介します．ここでの知識は，以降の章，特に４章・５章を理解する助けとなるでしょう．またPC-8801が持つ数々の優れた特徴もここでまとめて紹介していますから，実際に，PC-8801をお持ちでない方にとっては，重要な情報となるでしょう．

2章　BASICを知ろう

この章は，いわば予備知識にあたる章であり，N88-BASICというより，BASIC言語自体をとりあげ，BASICとは何か，BASICの性格はなどを解説しています.「BASIC言語を使うのは初めて」という方には，是非とも読んで頂きたい章です.

3章　プログラミングの基礎

2章に続いてBASIC言語について書かれたものですが，プログラミングということを主題においています.「BASICの個々の命令は理解できたんだが，うまくプログラムが組めない」という方への解説です.

4章　グラフィックス・ワールド

PC-8801の最も優れた機能の1つである，グラフィックスについての解説です．N88-BASICが持つ数々のグラフィック命令を中心として話を進めますが，実用プログラムを多用して理解の手助けとしています.

5章　入出力とファイル

この章では，N88-BASICで扱える入出力装置の紹介から，データファイルの作り方までを詳しく解説しています．特にデータファイルの項では，N88-DISK BASICを導入し，ランダムファイルの作り方までを解説していますから，〝ビジネスへの応用〟を考えている方にとっては是非とも必要な知識でしょう．

6章　割り込みとその使い方

PC-8801では，様々なハードウェアからの割り込みをBASICで扱うことができます. この章では，割り込みの概念から，それら様々なハードウェア割り込みを使いこなすテクニックを解説しています.

7章　うまいプログラミングのために

本当に使えるプログラムとは何か，よいプログラムを作るには，ということを中心に，これまでの章で紹介できなかった命令の解説も含めて，高度なプログラミング技法について述べています．今まで,「BASICはすべて分っている」と思っていた方もきっと新しい発見があることでしょう．

8章　うまいデバッキングのために

プログラミングに絶対といっていいほど付きまとうデバッギング．本章では完全なプログラムを作成していく過程において，このデバッギングのノウハウを公開します．7章と共に，うまい

プログラムを完成させる有用な情報となるでしょう.

このように，本書は，PC-8801の機能の紹介から，プログラミング技法について述べられている訳ですが，メインテーマは，あくまでN_{88}-BASICの解説にあり，紙面の大半をこれに当てるのをよぎなくされています．したがって，各内容において，深く堀り下げて解説しきれなかったものもあります．それらについては，次の書籍を参考にして頂ければ幸いです.

N-BASICについて	「N-BASIC入門」	アスキー出版
入出力とファイルについて	「Disk BASIC入門」	〃

1982年　4月　著者

☆ お知らせ ☆

アスキー出版では，本書に続く第2巻として「PC-8801グラフィックスのすべて」の発売を予定しております．内容は，〝図形の描き方〟〝グラフの作り方〟〝静的・動的シミュレーション〟〝アニメーション〟〝3Dグラフィックス〟〝漢字の応用〟などで，本書中で紹介しきれなかった機能と，その応用のすべてを公開します.

目　次

3章 プログラミングの基礎………61

4章 グラフィックス・ワールド………111

5章 入出力とファイル………185

7章 うまいプログラミングのために………251

8章 うまいデバッギングのために………273

1章 PC-8801のプロフィール

CRT
プリンタ
本体
ミニディスク(片面用)
キーボード

PC-8801は，今やベストセラー機種ともなったPC-8001の上位機種として発表されました．これは，8bit CPUによるパーソナルコンピュータの可能性を極めたとも言えるほど，機能を大幅に拡張していますが，ここではまずこのPC-8801の特徴をハードウェアの面から紹介しましょう．

1.1 ハードウェアにみる特色

1. 広大なメモリ容量

まず，何と言ってもユーザーが扱えるプログラム領域が，64Kbyteと大きくなりました．グラフィックエリア，ROM領域まで含めると，実に184Kbyteという大容量メモリが本体に実装されていて，さらに拡張する事で272Kbyteにもする事ができます．

PC-8801で採用されているCPUμPD780C-1（Z-80Aコンパチブル）で扱う事の出来るメモリ空間は64Kbyteですが，この大容量メモリをバンク切り替え機能により実現しています．

＊：N88-BASICモードでは，テキストエリアのIKバイトが変数エリアのメモリウィンドウに転送されて処理される．

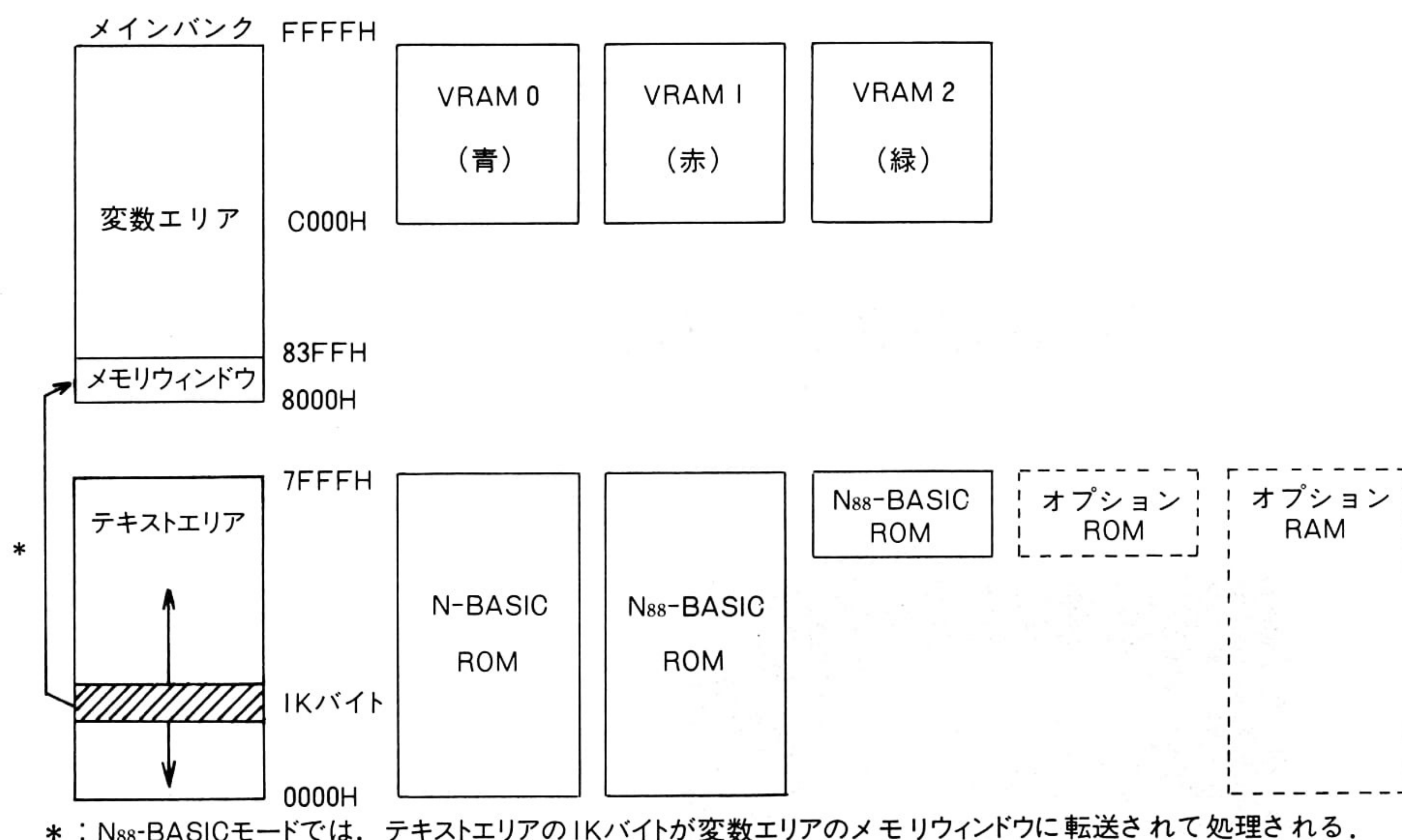



図1　メモリマップとメモリウィンドウ

また，BASIC 実行時において，ユーザーエリアを 64Kbyte とするために，図の様なメモリウィンドウをハードウェアによって実現しています．

2.　テキスト RAM とグラフィック RAM を個別に装備

今までのパーソナルコンピュータでは普通，文字を表示するキャラクタ画面と，LINE 文や PSET 文等で図形を表示するグラフィック画面とは同一の画面を指しており，そのため文字が表示されている所に直線を引いたりすると，その文字が消えてしまったり，また画面上の文字だけ，図形だけを消去する事が出来ませんでした．しかし PC-8801 では，この 2 つの画面を完全に分離したため，文字画面だけを表示したり，グラフィック図形だけを表示したり，さらにこの 2 つを合成する事が可能となりました．また，カラーディスプレイと白黒ディスプレイの 2 つを同時に接続

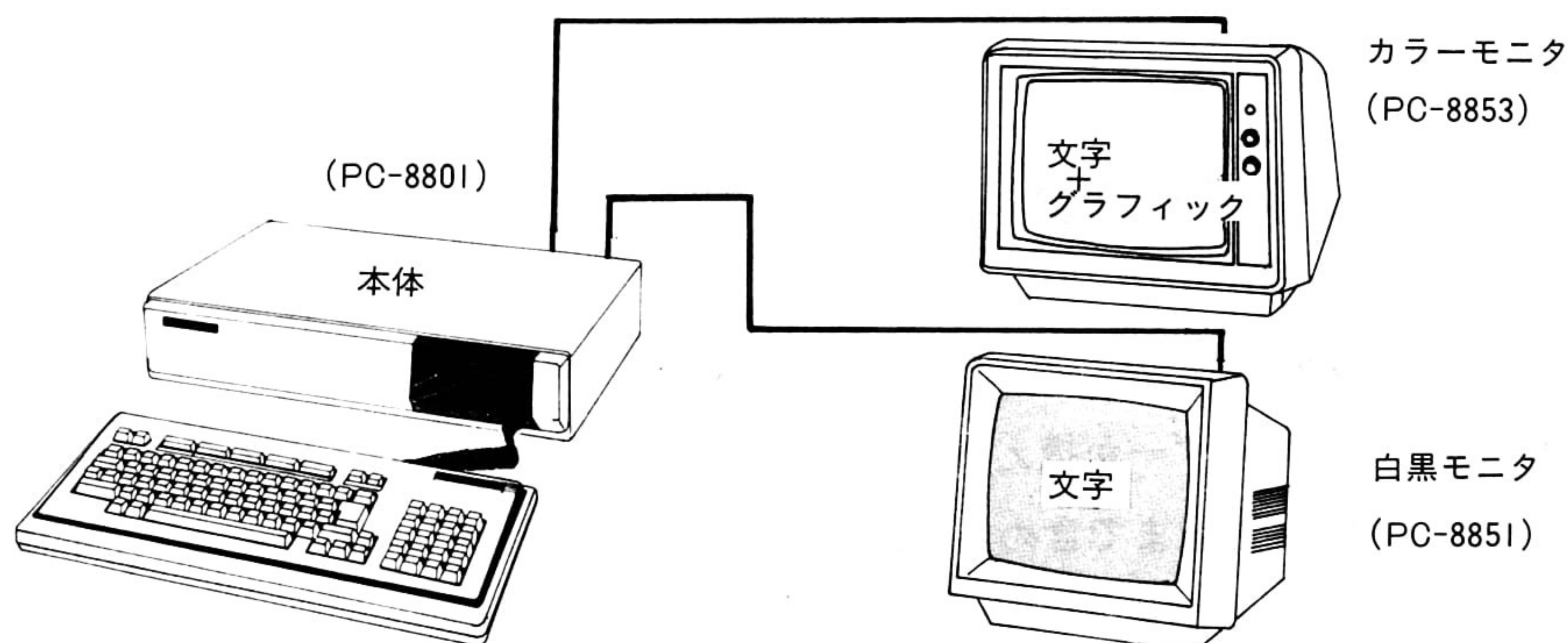



図2　2つの画面構成

した場合には，この2種類のディスプレイに表示する画面を，異なる画面の組み合わせとする事が可能です.

3. 3バンク48Kで構成されるグラフィックRAM

グラフィック画面用の領域は，16Kbyteずつの3バンク構成となっていますが，この3バンクを組み合わせる事で3種類の表示の方法が出来ます.

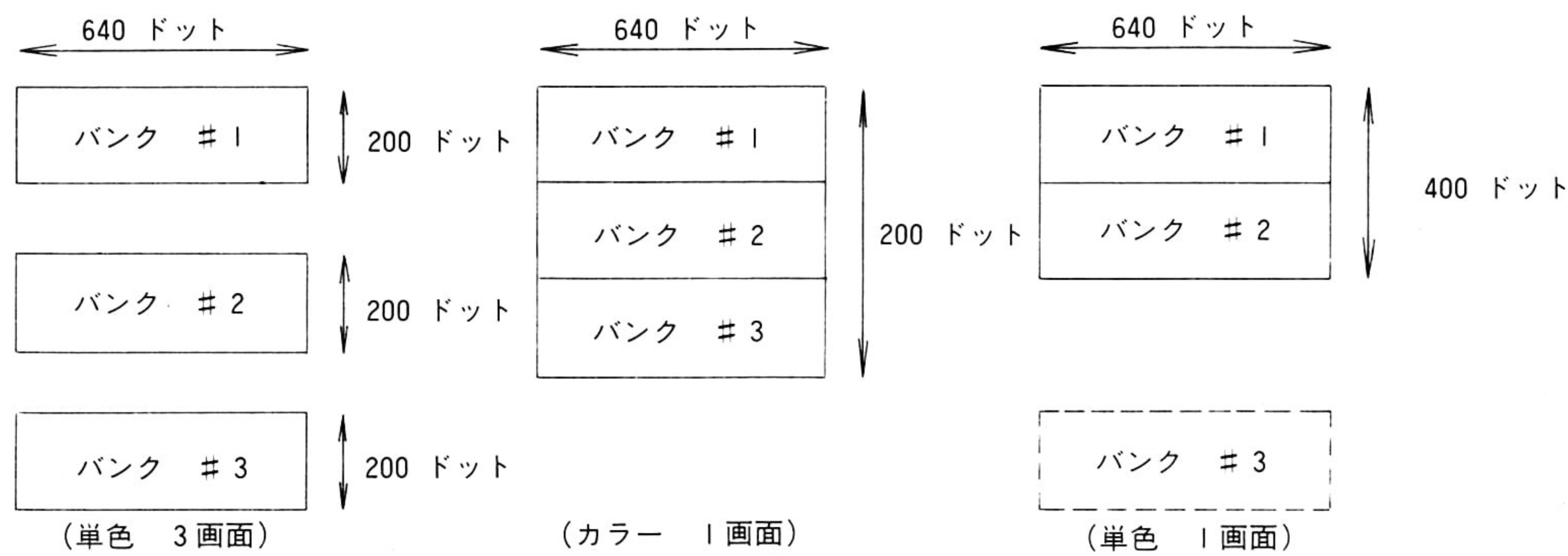



図3　3つのバンクと3つのモード

☆1バンクを使って白黒1画面を表示する

この場合には，画面のリゾリューションは640×200ドットとなりますが，このモードにおいては，一画面に1バンク費しますので3画面が確保できます．また，この時には表示画面と書き込み画面とをそれぞれ指定できるので，例えば3画面を合成して表示するとか，表示画面と書き込み画面を別にし，これを切り替える事で一瞬のうちに図形を描いたかのように表示する事が出来ます.

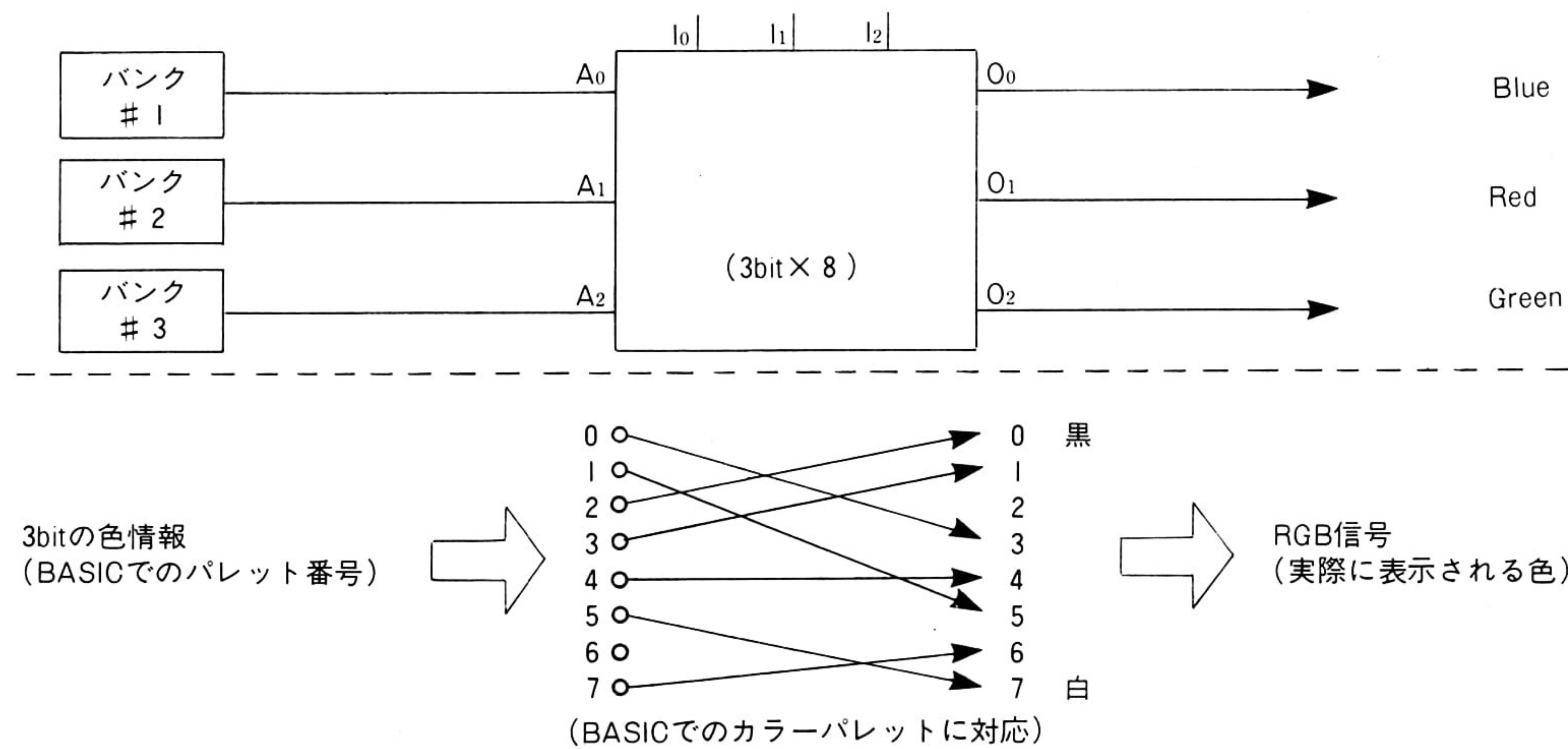



図4　カラーマッピング・RAM，3bitの情報と色信号との対応表

☆3バンクを使ってカラー1画面を表示する

この場合には，リゾリューションは 640×200 ドットで，1ドット毎に色を指定できます．この時に表示できる色は8種類ありますが，色の指定に関して特殊なハードウェアが付加されています．

一画面に3バンク使っているため，1ドットに対して 3bit の情報が得られますが，この 3bit をすぐに色信号とするのではなく，一度 マッピング・RAMという対応表を通して変換し，RGB 信号とするのです．そしてこのマッピング・RAMは書き替えられますので，3bit で表すカラーコードと実際の表示色との対応を自由にしかも高速に替える事が出来ます．

この事を利用すると，例えば，一度カラーコードを指定して画面に図形を表示すれば，次にその図形の色を変える場合でも，カラーコードを変えて作画しなおす必要はなく，マッピング・RAMを変化させるだけで実現出来ます．

☆2バンクを使って白黒一画面を表示する

この場合には，画面のリゾリューションは 640×400 ドットとなり，1ドットが画面上で正しく正方形となります．このモードは，主に漢字表示用として用意されたもので，一画面上に漢字パターンを40字×20行表示できます．ただし，このモードは640×400ドットの解像度を確保するために，ビデオ信号の同期周波数が異なる専用高解像度ディスプレイを使用しなければなりません．

また白黒モードのグラフィックの場合でも，キャラクタ画面と合成する事により，キャラクタ単位で図形に色を付ける事が出来ます．

4. 漢字 ROM をオプションで装備

本体内に，オプションで発売されている漢字 ROMボードをセットする事で，JIS 第一水準の漢字(2965種)と，非漢字(約700種)をグラフィック画面上に表示する事が出来ます．

5. 数種のディスクを接続可

フロッピーディスクの接続に関してはフレキシビリティがあり，8インチドライブ，5インチドライブのいずれでも接続でき，また混在して使う事も出来ます．

特に5インチドライブの場合には，インターフェイスがすでに本体内に内蔵されていますので，簡単に専用ケーブルで接続出来ます．

6. 電池バックアップのカレンダ時計を内蔵

カレンダ時計を内蔵しているので，いつでも呼び出して利用できます．またこれは Nicd 電池でバックアップされていますので，電源 OFF した場合でも，設定した日付，時刻が失なわれる事がありません（詳しくは，付属の PC-8801 ユーザース・マニュアルを参照してください）．

7. その他数種の周辺装置をサポート

フロッピーディスクの他に，数種の有用な周辺装置をサポートしています．

① **オーディオカセット**……一般のカセットテレコを手軽な外部記憶装置として使用できます．

② **プリンタ**…………………セントロニクス規格のプリンタを即使用可能です．

③ **RS-232C ポート**………標準規格のコミュニケーションポートを実装していますので，このポートをもつ他の機器とデータ通信ができます．

④ **ライトペン**………………ライトペンインターフェイスを本体内に標準実装しており，オプションのライトペンが利用できます．

8. N_{88}-BASIC と N-BASIC を搭載

この様に，PC-8801 のハードウェアは，数多くの特徴を持っています．この PC-8801 のハードウェアをソフトウェアからの観点で考えた時に，PC-8801 は PC-8001 に対して上位コンパチとなっています．

そこで，PC-8801 に ROM 化して内蔵されている BASIC は，PC-8801 の機能を最大限に発揮する N_{88}-BASIC の他に，PC-8001 の N-BASIC があります．

本体後側の DIP スイッチにより起動時に実行する BASIC を選択出来る様になっていますが，この N-BASIC は，あくまで PC-8001 のユーザーが開発したソフトウェアを利用するためのものですから，通常は N_{88}-BASIC を使ってください．

1.2 ソフトウェアにみる特色

次に，ソフトウェアの観点から見た，PC-8801 の特色を解説しましょう．

1. 2つの BASIC

ハードウェアの特色の所で述べたように，PC-8801は，N_{88}-BASIC と N-BASIC の 2 つのプログラム言語を持っています．しかし，N-BASIC は，PC-8001 用に開発・蓄積されている豊富なソフトウェアを利用するためであり，メインのプログラム言語は，あくまで N_{88}-BASIC です．

☆N_{88}-BASIC で新しく追加された命令

N_{88}-BASIC は，N-BASIC に様々な拡張命令を施したものですが，まずどのような命令が新しく追加されたかを見てみましょう．次に示す各命令は，N-BASIC にはなかった命令です．

コ　マ　ン　ド	NEW ON		
一　般　命　令	INPUT WAIT OPTION BASE WRITE	LINE INPUT WAIT RANDOMIZE SEARCH	WHILE～WEND
画面・グラフィック命令	CLS CIRCLE VIEW	SCREEN PAINT ROLL	COLOR @ WINDOW MAP
入出力命令	WRITE # ON HELP GOSUB STOP ON/OFF/STOP ON TIME$ GOSUB ON COM GOSUB PEN	HELP ON/OFF/STOP ON KEY GOSUB COPY TIME$ ON/OFF/STOP ON PEN GOSUB CHAIN	KEY(n) ON/OFF/STOP ON STOP GOSUB WIDTH LPRINT COM ON/OFF/STOP PEN ON/OFF/STOP COMMON
特　殊　命　令	CALL		

表1　N_{88}-BASIC拡張命令表

☆N-BASIC との主な相違点

N_{88}-BASIC は，N-BASIC に，PC-8801 が持つハードウェアの全てを活用するために，幾つかの拡張命令が付加されたもので，原則的にはコンパチビリティを持っています．しかし，N_{88}-BASIC には，ハードウェアの変更や BASIC の改良により，なくなった機能や内容の変わったものが幾つかあります．

次に示すのは，N_{88}-BASIC ではなくなった命令です．

CLOAD/CLOAD？/CSAVE

MOUNT/REMOVE/FORMAT

また，なくなった機能としては次のものです．

アトリビュート方式のセミグラフィック機能

キャラクタによって線を引く機能

さらに BASIC の内部的な処理の変更によりその内容が異なりました．

変数名の識別………………N_{88}-BASIC では40文字までの名前を区別しますが，N-BASIC では最初の2文字しか判別の対象としません．

数値の自動型変換の違い…実数値を整数値に変換する場合に，N_{88}-BASIC では小数点以下四捨五入，N-BASIC では小数点以下切り捨てます．

☆N_{88}-BASIC から N-BASIC への移行

PC-8801 には，N_{88}-BASIC と N-BASIC の2種類の異なる BASIC があります．

この2つの BASIC 間を移行するために，N_{88}-BASIC には NEW ON 命令があり，これによ

り N_{88}-BASIC から N-BASIC に変更できますが，この逆は，リセットまたは再起動にしてのみ行えます．

NEW ON 1……N_{88}-BASIC から N-BASIC へ

しかし，この移行の際にBASICは内部の初期設定をしますので，N_{88}-BASIC で N-BASIC のルーチンを呼び出す事はできません．さらに，内部で扱う BASIC の中間コードが異なるので，カセットやディスク等にセーブされたそれぞれのプログラムを，BASIC を変えてロードする事は簡単に出来ません．

ディスクをお持ちの方の場合は，N-BASIC のプログラムをアスキーセーブしておけば，N_{88}-BASIC でも読むことができます．しかし，N-BASIC と N_{88}-BASIC では，グラフィック命令などにおいて，全くコンパチブルという訳にはいきませんから，多少の変更は必要となります．

2．豊富なグラフィック命令

まず，何と言っても PC-8801 では，グラフィック機能が強化されていて，N_{88}-BASIC には，これをサポートする種々の強力な命令があります．ハードウェアの特色で述べた様に，グラフィック画面は 3 種類のモードがあり，このモードを切り替える命令が当然ありますが，モードによってリゾリューションが違います．

そこで，グラフィック命令でこれに対して統一的に対応し，またグラフィックを使うプログラムを簡略化するために，論理的な座標系の考え方を導入しています．

WINDOW 文によるワールド座標系と，VIEW文によるスクリーン座標系がそうです．

ワールド座標系とは BASIC で扱う事のできる数を範囲とした論理座標系で，この座標系内の任意の位置を指定して，実際に表示する領域を指定する事が出来ます．グラフィック命令の多くがこの座標系を使いますので，これにより，画面上の位置を気にせずに，柔軟な位置指定が出来ます．

また，VIEW 文によって画面上の任意の領域を表示領域（VIEW ポート）に指定できますが，この VIEW ポート内の座標系が，スクリーン座標系です．

と言っても理解できないかもしれませんが，詳しくは 4 章のグラフィックス・ワールドで説明します．

また，漢字 ROM のための文として，PUT 文が拡張されたり，ハードウェアの特徴の所で述べた，カラーマッピング・RAM をカラーパレットという考え方のもとに，COLOR 文が強化されています．

3．すべての入出力機器をファイルとして統一

PC-8801 では，キーボード，ディスプレイ，フロッピーディスク，RS-232C，オーディオカセット等多くの入出力機器がありますが，これらを簡単に統一的に扱うために，『ファイル』の考え

方を導入しました．このため，プログラムで入出力を行う場合には，機器によって使う命令を選ぶ事なく行える様になりました．

4．使い易い変数名とラベル名

また，変数名は40文字まで許され，しかもN-BASICの様に頭の2文字だけを比較するのではなく，40字すべてを比較します．

さらに，GOTO文等のプログラム中の飛び先の指定として，ラベル名を使う事ができ，このため行番号を使う事なく，ラベルによりプログラムが分かり易くなり，プログラムの開発やデバックの効率が向上しました．本書のサンプルプログラム中でもこのラベル名を数多く使っています．

以上のように，様々な機能がインプリメントされています．この他にも，面白い機能，有用な機能がありますが，それらは各章で詳しく紹介していくことにしましょう．

2章 BASICを知ろう

2.1 PC-8801を電卓として使う

皆さんの中には，パソコンについては何も知らないけれど，電卓なら大丈夫という方も多いのではないでしょうか．そこで，ここではPC-8801を電卓風に使ってみて，皆さんにN_{88}-BASICの文法などの基礎的な知識をつけて頂き，併せて文字の入力などキーボードの操作を習得して頂くということを，多少欲張りすぎかもしれませんが，目標として話を進めていきます．

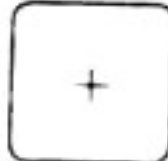
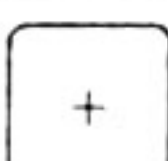
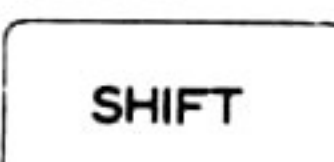
ところで最近の電卓の進歩（進歩と言うよりむしろ変化と言った方がよいかもしれませんが）は大変なものです．関数やメモリが付いたものはともかくとして，ゲームや占いができるものまで登場しています．実に様々な種類の電卓があるものです．

さてここでは，四則計算と幾つかの関数，そしてメモリについての機能をPC-8801で行わせてみることにします．N_{88}-BASICをもってすれば，現在ある電卓の全ての機能を実現させることも可能であることが，段々と皆さんにも分かってくるのではないでしょうか．

2.1.1 電卓とPC-8801

電卓とPC-8801を見比べてみると，外観がかなり違っています．あえて似ている所を探せば，PC-8801のキーボードの右側のテンキーの部分でしょう．しかし，違って見える各部分も，その機能，働きを考えると，同じ働きをするものだと分かります．つまり，電卓もPC-8801も全体を3つの部分に分けて考えることができるのです．それはキーボード，本体，表示部（ディスプレイ）の3つです．それぞれの働きを説明すると，キーボードはオペレータが数字や文字を入力するためのもの，本体はキーボードから入力されたものを理解し仕事をするもの，ディスプレイは本体が行った仕事の結果などをオペレータに知らせるために，文字や数字などを表示するためのものです．このように，どちらも似たような仕組みを持っています．

これは，実はどちらもマイクロコンピュータを内蔵したコンピュータだからなのです．電卓がコンピュータと聞いて驚かれる方も多いかもしれませんが，マイクロコンピュータは，もともと

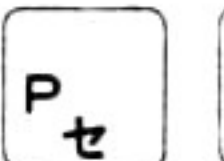
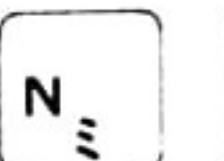
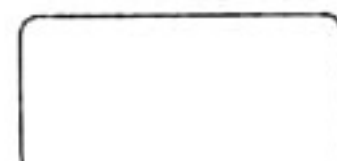
は電卓のために開発されたものです．皆さんは，知らない内にコンピュータと長い間，身近に付き合ってきたのです．

ところが，この2つはどちらもコンピュータなのですが，その機能は大きく異なっています．それは，内蔵されたマイクロコンピュータが，全く別の働きをするように，プログラミングされているからなのです．電卓のマイクロコンピュータは電卓の働きをするように，またパソコンのマイクロコンピュータはパソコンの働きをするようにプログラミングされているのです．そして，その働きに合うように，キーボードや表示部が備えられているのです．

ここで言っているプログラミングという単語の意味は，コンピュータに仕事の手順を覚えさせることです．つまり電卓というコンピュータに電卓として働くように，その仕事の手順を記憶させるのです（プログラミングについては後に詳しく説明します）．

このPC-8801は，パソコンとして働くようにプログラミングされており，それによって私達はN_{88}-BASICという言語を使うことができるようになっているのです．PC-8801は，N_{88}-BASICで書かれた仕事を理解し実行します．ですから皆さんは，N_{88}-BASICを用いて，プログラミングしなければならないのです．それではキーボードから何か入力してPC-8801に仕事をさせてみましょう．

2.1.2 まずは直接モード(ダイレクトモード)から

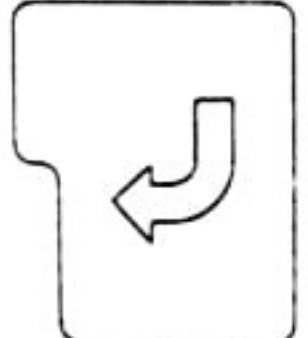
PC-8801にキーボードから何か入力する前に，電源が入っており，画面にOkという表示が出て，その下で四角いカーソルが点滅しているか見て下さい．この状態は，PC-8801側の準備ができて，オペレータからの入力を待っている状態です．これをコマンド待ちの状態，またはコマンドレベルと呼んでいます．

それでは電卓風に何か計算させてみましょう．1＋2を計算させてみます．電卓では，[1] [+] [2] [=] と押せば，答が3と表示されます．さっそくPC-8801でもやってみましょう．[1] [+] [2] [SHIFT] ＋ [= - ホ] とキーを押してみます．数字と＋は右側のテンキーを使うと分かり易いでしょう．＝はシフトキーを押しながら [= - ホ] のキーを押します．押したキーの文字は画面にそのまま表示されていきます．現在のカーソルの位置に入力された文字が表示され，カーソルはその文字のすぐ右に移っていくのです．

```
Ok
1+2=■
```

残念ながら，＝を入力しても答は出て来ませんでした．このままほおっておいても答は得られません．どうもPC-8801は，私達の入力したものを理解していないようなので，少し変えて入力してみましょう．

まず [INS DEL] のキーを押して下さい．

```
Ok
1+2■
```

カーソルが１つ左へ戻っていきました．もう３回押すと，入力した文字が全部消えてしまい，カーソルは以前あった場所（Ok の O の下）に戻ってしまいます．このようにこのキー（デリートキー，DEL キー）は，カーソルの１つ左の文字を消す働きがあるのです．もしキーを押し間違えたり，入力する文字を勘違いした場合などは，このキーを使ってその文字を消して，正しい文字を入力しなおせばよいのです．

今度は，print　１＋２と入力して下さい．キーは [P セ] [R ス] [I ニ] [N ミ] [T カ] [　　] [1] [+] [2] と押します．[　　] はキーボードの一番下側にある長いのっぺりしたキーで，スペースキーと呼びます．これは空白（スペース）を入力するためのキーです．

```
Ok
print 1+2■
```

このように表示されます．もし間違ったならば，デリートキーで訂正します．前と同じように，このままでは何も起りません．そこで今度は [⏎] というキーを押してみます．

```
Ok
print 1+2
 3
Ok
■
```

どうですか．PC-8801 は３という答を表示して，コマンド待ちの状態になりました．この print という単語は，後に続く計算式の結果を表示する働きを持つ，BASIC の命令の１つです．そして [⏎] は，その命令を実行させるものだったわけです．この [⏎] キーのことを，リターンキーと呼んでいます．

[⏎] ＝ [⏎] ＝ ⏎

これまでの実験で，電卓と BASIC の違いがはっきりしてきました．

① PC-8801では，オペレータが入力したものを BASIC に理解させるためには，終りにリターンキーを必ず押さなければならない．リターンキーを押す前ならば，オペレータは今までに入力した文字を訂正することができる．

② BASIC では，BASIC 言語を用いて実行させる仕事の内容を記述しなければならない．画面に計算結果を表示させたい場合は，画面に表示を行わせる命令（PRINT のことです）を用いて計算式の答を表示させる必要がある．

以上の２つの事柄をしっかりと心の中に留めておいて下さい．

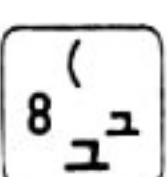
それでは，BASIC を電卓の代わりとして使ってみましょう．まずは四則計算です．

2.1.3 四則演算は今まで通り

四則演算とは，たし算，ひき算，かけ算，そしてわり算のことです．数学ではそれぞれの演算を，＋，－，×，÷の記号で表していますが，BASIC では加減算の記号は同じですが，乗算は＊，除算は／の記号で表します．違っている点はそれだけで，それ以外は，普通の計算と変わりません．乗除算を加減算よりも優先して行うことなども同じです．

それでは何か計算してみましょう． [P] [R] [I] [N] [T] [ ] [1] [+] [2] [*] [3] [↵] と入力して下さい．

```
print 1+2*3
 7
Ok
■
```

これは１＋２×３を計算するものです．計算の結果を表示する方法が分かりましたか．今度は次の計算を実行してみて下さい．

１＋３÷２，２÷２÷２，１×２－３

```
print 1+3/2,2/2/2,1*2-3
 2.5            .5            -1
Ok
■
```

ちょっと例をよく見て下さい．それぞれの計算式が，（コンマ）で区切られて，PRINT の後に続いています．こうすると計算結果を適当な間隔をおいて表示してくれるのです．これは表示したい結果が幾つかある場合に，１つの PRINT で済ませることができるので便利です．

次にカッコ（　）を使って計算を行ってみましょう．BASIC はカッコも使えるようになっているのです．次の計算を実行してみて下さい．

２×（１＋３），１－３÷（２＋１）．

カッコを入力するには [SHIFT] ＋ [8 (] のようにシフトキーを押しながら目的のキーを押します．

```
print 2*(1+3),1-3/(2+1)
 8              0
Ok
■
```

カッコを使うと，その中の計算を一番に行います．ですから，乗除算より加減算を先に行わせることができるのです．また，カッコの中にカッコがあるような使い方をすると，一番内側のカッコの中から計算を行います．次の計算を実行してみて下さい．

８÷（２×（３＋１））

```
print 8/(2*(3+1))
 1
Ok
■
```

カッコを使えば計算式通りに入力できますから，電卓より便利です（最近の電卓にはカッコを使えるものもありますが）．

また BASIC では数値に符号を付けることもできます．符号とは－２とか＋３のように数値の正負を表す記号のことです．通常は＋の符号は省略します(＋３は３と同じ意味です)．では次の式の値を求めてみて下さい．

２＋－１，２×－１，－（－１）

```
print 2+-1,2*-1,-(-1),-(-(-1))
 1             -2             1             -1
Ok
■
```

結果を見ると，－の符号はもとの値の符号を反転（－を＋に，＋を－に）する働きがあることが分かります．

さて，これまで四則演算についていろいろ試して来ましたが，分かって頂けたでしょうか．基本的なことは，BASIC の演算のルールが，数学的な（つまり学校で習ったような）演算のルールと同じであることです．とは言っても，×の代りに＊を使うなどの表記上の違いは多少ありますが，しかし，数学的な式を BASIC での表記法に変えることは簡単にできるでしょう．2, 3 例をあげておきます．

数学的記法 ──────→	BASICの記法
２＋3.5÷（－４）	２＋3.5／（－４）
－［８－２×｛３－（－２）｝］	－（８－２＊（３－（－２）））
$-\frac{2+3}{4-5}$	－（２＋３)/（４－５）

皆さんもいろいろな計算を BASIC で行ってみて下さい．

なおこの時，ピーと音がして何かメッセージが表示される事があるかもしれません．これはオペレータ（あなたのことです）が入力したものに，基本ルールに合わないところがありますよ，と BASIC が返答をして来たのです．こんな時は，入力したものをもう一度よく見てみましょう．どこか間違っているはずです．例えば次の様な場合，

```
primt -1/2
Syntax error
Ok
print -1/(2-2)
Division by zero
-1.70141E+38
Ok
■
```

最初のは print を primt と間違えています．２番目のは０で割り算を行おうとしています．これは数学的に不可能な演算です．このように，BASIC は入力されたものに不合理な点があれば，

間違ってますよとオペレータに知らせてきます．これをエラーメッセージと呼んでいます．詳しくは後の章で説明しますので，今の所はエラーメッセージが出たら，どこか間違えたな，と思って下さい．

2.1.4　その名はM(メモリ)

皆さんは，電卓に付いているメモリ機能を御存知ですか．これは計算結果などの値を記憶させておける機能です．そして，後になってその値を取り出して，他の計算に使ったりすることができます．最近のたいていの電卓には，最低1つのメモリが備えられています．中には数個から数十個ものメモリを持ったものもあるのです．

ところで，BASICにもこの便利なメモリに相当する機能があります．それは変数と呼ばれるものです．さて変数という言葉を聞いて，皆さんは何を連想しますか．代数学，それとも方程式ですか．数学の世界にも変数の考え方は登場しますが，BASICでの変数も同じような概念です．つまり変数とは，数値を表す文字で，その中味の値がいろいろ変えられるものと考えて下さい．変数を数値の入れ物と考えてもよいでしょう．

例えばAという変数があるとします．そしてその変数の表す値が3だとします．これは，変数という箱の中に3という数値が入れられており，その箱にはAという名前が付いていると考えることができます．このAという名前を使えば，箱の中の3という値を取り出すことができるのです．

変数の考え方

［Aという変数］

A

中に入れる数値を自由に変えることができる（代入）

数値の入れ物(変数)

入れ物の名前(変数名)

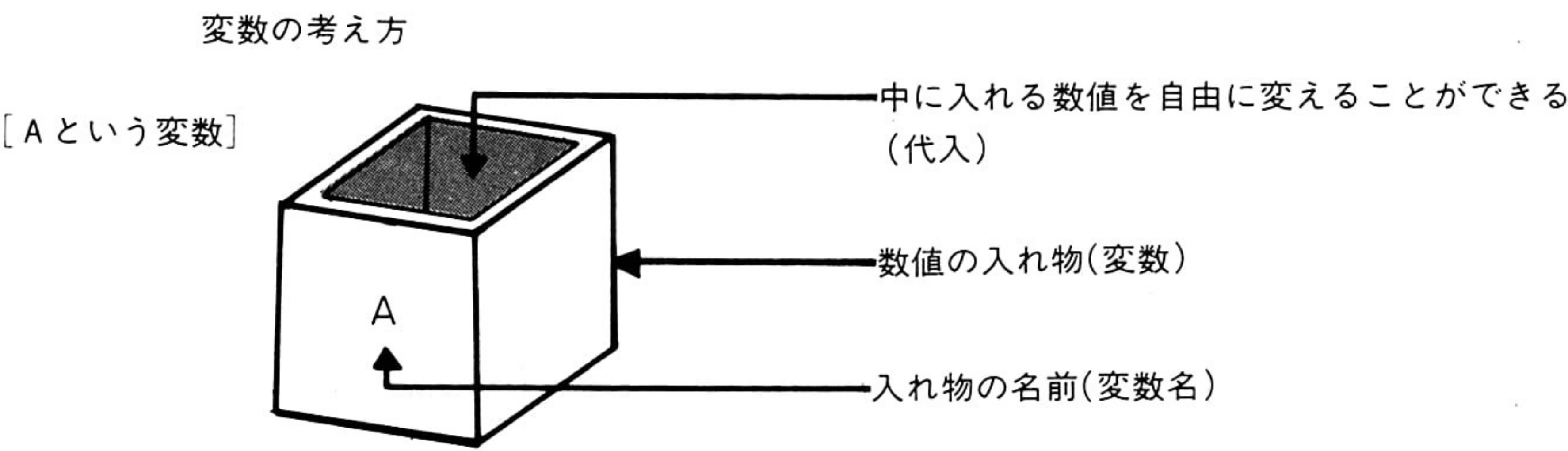



図1　変数の考え方

では次の様に入力して下さい．

```
a=3
Ok
■
```

これはAという変数の箱に，3という値を入れる手続きです．このことを変数Aに3を代入すると言います．そしてこの手続きを行う文を代入文と呼びます．

これでAの値が3になりました．本当かどうか次にAの値を表示してみましょう．次の文を入力して下さい．

```
print a
 3
Ok
■
```

PRINTは変数の値を表示させることもできます．ここではAの値は3であると表示した訳です．

Aは変数ですから，値を変えることができなければなりません．変数は，代入文によってのみ変えることができます．それでは，Aの値を変えてみましょう．次の代入文を入力して下さい．

```
a=a+2
Ok
■
```

続いて，Aの値を表示してみます．

```
print a
 5
Ok
■
```

どうです，値が変りましたね．ところでこの代入文はどういう意味でしょうか．これはAの値に2を加えた結果をAに代入せよという事です．先程（この代入文を行う前）のAの値は3でしたから，2を加えて5になります．この値がAに代入された訳です．

ところで皆さん，この例を見て何か変な感じがしませんか．A＝A＋2という代入文の＝を，イコール，等しいの意味に捕えると大変です．代入文の等号は数学的に等しいの意味ではありません．右辺の値を左辺の変数に代入するの意味なのです．つまり右辺のAは変数Aの中味である値を示し，左辺のAは変数A（Aという箱）そのものを示しているのです．

変数はある値を示すものですから，これを計算式の中に入れて計算を行うこともできます．

```
print a+2
 7
Ok
■
```

計算を行った結果を変数に代入したければ，代入文を使えばよいのです．今度は新しくBという変数を作って，それに代入してみましょう．作るといっても特別な手続きが必要な訳ではなく，式や命令の中に今までになかった変数を使うと，BASICが自動的に新しくそれ用の変数を作ってくれます．

```
b=a+5
Ok
print b
 10
Ok
■
```

さてＢという変数が登場しましたが，これによって，もとからあった変数Ａに何ら影響を及ぼす訳ではありません．

```
print a,b
 5             10
Ok
■
```

変数を表す名前（ここではＡとＢ）が違うため，この２つは全く別のものなのです．

皆さんは変数が使えるようになりましたか？ 電卓に付いているメモリ機能は，あれば便利というぐらいの物ですが，BASICにおける変数は必ずなくてはならない大切な概念です．これからも度々出て来ます．重要な事柄ですから，後でもう一度詳しく説明します．

2.1.5 関数を使って

関数と言われても，あまり縁の無い方が多いのではないでしょうか．「函数なら知ってるぞ」と言う方も，もしかしているかもしれません．技術関係の方なら身近に聞く言葉でしょう．関数電卓という強い味方がありますから．BASICにも，関数電卓ほどの種類はありませんが，幾つかの関数が用意されています．サイン，コサイン，タンジェント，ルート，ログ，………etc．ここでは一例としてスクウェアード・ルート（平方根）を使ってみます．

```
print sqr(2),sqr(4)
 1.41421       2
Ok
■
```

SQRというのが平方根の関数名です．もちろん，その他の関数にもそれぞれ名前が付けられています．関数名に続くカッコで囲まれた数値の，関数の値が求められます．前に出て来た四則演算と組み合せて用いることももちろんできます．

```
print 1+sqr(6-4/2)
 3
Ok
■
```

もう１つ，紹介しましょう．皆さんは絶対値という言葉を御存知ですか．簡単に言えば負の数を正の数に変えることです．この絶対値を求めるのが，ABSという関数です．

```
print abs(-12.3)
 12.3
Ok
print sqr(abs(-4))
 2
Ok
■
```

これまで電卓の機能をモデルとして，BASICの使い方を説明して来ましたが，皆さんはBASICを使うことに慣れてもらえたでしょうか．ところで，これまで行ってきたような BASICの使い方は，直接モードによる命令の実行と呼ばれています．

この直接モードとは，オペレータが入力した命令（代入文やPRINT文）をその場ですぐに実行する動作のことです．今までにBASICで実行して来たことは，全部すぐにその場で実行して，即座に結果を得ることができましたね．

この直接モードで命令を実行することは，すぐに結果が求められる便利さはありますが，反面，同じようなことを何度もやらせたい場合は，その都度入力しなければならないので不便です．例えば，あなたがローンの計算をしたいという場合，頭金，手数料などの条件を変えて月々の支払いがいくらになるか求めるにも，それぞれ条件を変えた計算を，いちいち入力しなければならないのです．こんな時，最初の条件だけ入力すれば後はコンピュータが自動的に結果を求めてくれると大変便利です．これを実現するには，コンピュータに計算の手順を覚えさせておければいい訳です．

それでは，コンピュータをもっと便利によりコンピュータらしく使える，この方法についてお話ししましょう．

2.2 間接モード(プログラムモード)で使う

コンピュータのコンピュータたる最大の理由は何だと思われますか．それは，与えられた命令の列を記憶し，そしてそれを実行する能力です．命令の列というのは，実行する仕事，処理の手続きが示されているもので，プログラムと呼ばれています．このプログラムによって，コンピュータにさせたい仕事を覚えさせておけば，後で何度でもその仕事を行わせることができます．しかもその仕事ぶりは，速く，そして正確です．これは電卓には無い機能で，コンピュータだけの特徴です．電卓ではその都度キーを押して指令を送ってやらなければなりませんでした．間接モードとは，命令をその場で即実行するのではなく，プログラムとして一旦内部に記憶し，後で必要な時に順に実行する所からきています．このプログラムを作ることを，プログラミングと呼びます．

2.2.1 プログラムを作る

それでは BASIC の命令をその場で実行するのではなく，プログラムとして記憶させるにはど

うすればよいかといいますと，命令の前に番号を付けて入力すればよいのです．例えば，PRINT 1＋2を 100 PRINT 1＋2として入力するのです．

PRINT 2	──────→100 PRINT 1+2
その場で実行される．	**実行は行われずプログラムとなる．**
（直接モード）	**（間接モード）**

番号を変えて別の命令を入力すれば，それも記憶されます．たくさん命令を入力して，後でそれらをまとめて実行させることができるのです．ところでたくさんの命令を入力してプログラムにする場合に重要なことが一つあります．それは番号の付け方です．番号の付いた命令がたくさん入力されている場合，BASICがそれを後で実行する時は，番号の若い順に実行していくのです．つまりこの番号は，プログラムの実行順番ということになります．この番号は行番号と呼ばれて，PC-8801では1から65529までの整数を使うことが許されています．

それでは，実際にプログラムを入力してみましょう．次の各行を入力して下さい（行の終りでは，リターンキーを押すことを忘れないようにします）．

```
100 A=4
110 PRINT 2*(A+3)
120 END
```

各行の先頭にある番号はもちろん行番号です．入力した時に，BASICの反応が今までとは違うことに気付いたと思います．つまりOkの表示が出ないのです．しかしコマンドレベルであることには変わりありません．ただ入力した行が，プログラムとして記憶されただけなのです．ちょっと行番号120の行（これを120行と呼びます）を見て下さい．ここにENDという命令が入っています．これはプログラムの実行をここで終らせるための命令です．後でBASICがプログラムを実行した時，このENDにくるとそのプログラムを終了して，Okの表示と共にコマンドレベルに戻ります．

2.2.2 プログラムを実行する

それではこのプログラムを実行させてみましょう．それにはRUNという命令を使います．RUNと入力してみて下さい．

```
run
 14
Ok
■
```

プログラムが実行され，コマンドレベルに戻りました．ところで，プログラムは記憶されている訳ですから，何度でも同じように実行することができます．しかし，この程度のプログラムならば，何度も実行する必要はありませんし，ダイレクト（直接）モードで計算を行えば十分でし

ょう．けれどもコンピュータに行わせる仕事が複雑になると，プログラムを作った方が，仕事の能率がよくなってきます．というよりプログラムなしには不可能になってきます．プログラムを作るには手間がかかりますが，その手間に見合うだけの，いえそれ以上の利点，仕事を行う速さと正確さを，コンピュータは私たちに与えてくれるのです．

2.3 プログラムを編集してみよう

次はプログラムの編集についてお話しします．これまでの話で，プログラムの入力の仕方はお分かりでしょう．しかし，プログラミングは，それだけではできません．現在入力されているプログラムの内容を調べたり，新しいプログラムを追加したり，古いプログラムを変更，削除したりできなければ，プログラムを作る作業が思うようには進みません．プログラムの編集とは，プログラムの入力，追加，変更，削除などを行うことです．

2.3.1 プログラムを表示する

現在入力されているプログラムを表示させるには，LIST 命令を使います．LIST と入力してみて下さい．

```
list
100 A=4
110 PRINT 2*(A+3)
120 END
OK
■
```

こうすると，プログラムを行番号の若い順に表示してくれます．この表示された内容を，プログラムリストと呼んでいます．そしてプログラムリストを表示させること（LIST 命令を使うこと）を，一般的には〝リストをとる″，と言います．

ところで，リストをとったプログラムと以前に入力したプログラムを比べてみて下さい．PRINT や変数名など，小文字で入力したはずのものが，大文字で表示されていませんか．BASIC は，命令や変数名などにおいて，英文字の小文字と大文字を区別していません．プログラムの入力の際には，小文字と大文字のどちらで入力してもよいのです．それがリストをとった時には，命令や変数名などは，すべて大文字に統一されて表示されます．

もし，あなたが小文字で入力するのがいやで，いつも大文字で入力したい場合は，キャピタルロックの機能を使うとよいでしょう．これは通常の状態で，シフトキーを押しながらアルファベットのキーを押して，大文字を入力していたのを，そのままキーを押せば大文字が入力できるようにするものです．キーボードの左のシフトキーの上にある CAPS というキーを押してみて下さい．キーが少し下がってロックされたでしょう．この状態でキーを押せば，大文字が入力できるのです．もし小文字が入力したい時は，シフトキーを押しながらキーを押せばよいのです．この機能

を解除するには，もう一度キャピタルキーを押します．この機能は皆さんの好みに応じてどちらでも……といったところでしょうか．

話が横にそれましたが，LIST 命令は，先程の例のように，プログラムの全ての内容を見る他，プログラムの任意の行を表示することもできます．例えば任意の1行を表示したいのであれば，その行番号を LIST の後に1つ以上の空白を空けて指定します．

```
list 110
110 PRINT 2*(A+3)
Ok
■
```

また任意の範囲を表示したいのであれば，〝始点行番号－終点行番号〟という形で指定します．

```
list 110-120
110 PRINT 2*(A+3)
120 END
Ok
■

list -110
100 A=4
110 PRINT 2*(A+3)
Ok
■

list 110-
110 PRINT 2*(A+3)
120 END
Ok
■
```

その他，代用記号のピリオド（.）を使うと便利です．このピリオドは，LIST 命令の他，後に出てくる DELETE, EDIT などにも使うことができる代用記号で，現在 BASIC が着目している行番号の代用としての働きがあります．BASIC が現在着目しているとは，実行が中断された行，エラーが発生した行，変更された行など BASIC の現在のまたは，直前の実行の対象だった行を言います．これを使うとエラーが発生した場合など，即座にその行を表示することができます．

```
list 110
110 PRINT 2*(A+3)
Ok
list .-
110 PRINT 2*(A+3)
120 END
Ok
■
```

2.3.2 プログラムを変更する

次にプログラムの内容を変えたいときはどうすればよいか，お話ししましょう．ちょっと次のプログラムを見て下さい．

```
100 INPUT A
110 PRINT A,A*A,SQR(A)
120 END
```

INPUT という新しい命令が出てきました．これはキーボードから入力された値を，変数に代入するための命令です．INPUT　A は，変数 A にキーボードから入力された値を代入するのです．そして，プログラムでは，次に A の値の 2 乗と平方根を計算し，結果を表示しています．今度はこのプログラムを入力してみましょう．

まず，今あるプログラムのリストをとってみて下さい．新しく入力するプログラムと見比べると，100行と110行が違っていることが分かります．この行を変更すれば最も簡単です．このように，現在ある行の内容を変更したいときは，同じ行番号を使って新しい行を入力します．では，実際に変更してみましょう．

```
list                          ←─リスト命令の入力
100 A=4                       ┐
110 PRINT 2*(A+3)             │リストの出力
120 END                       ┘
OK
100 input a                   ┐
110 print a,a*a,sqr(a)        ┘変更する行の入力
■
```

では，リストをとってみます．

```
list                          ←─リストをとる
100 INPUT A                   ┐
110 PRINT A,A*A,SQR(A)        │変更されたプログラムのリストの出力
120 END                       ┘
OK
■
```

100行と110行が，確かに変更されていることが分かります．では，変更されたプログラムを実際に，実行してみましょう．

```
run   ←─RUN命令の入力
? ■   ←─入力待ち
```

? が表示されカーソルが点滅しています．これは INPUT 命令を実行して，キーボードからの入力を待っている状態です．そこで何か適当な数値を入力します．入力は数字を押して終りにリターンキーを押します．このように INPUT 命令は，キーボードからの入力が終了するまでは次の命令に進みません．

```
run
? 2
 2              4               1.41421 }計算結果の表示
OK
■
```

入力された数値は，変数Aに代入されて計算が行われました．別の値で計算したい時は，もう一度 RUN 命令で実行すればよいのです．

それでは，もう少しプログラムに手を加えてみましょう．今度は，AとBの2つの変数に値を代入して，その合計と平均を求めてみます．そこで，プログラムを次のように変更します．

```
105 INPUT B              ←—変数Bへの値の入力
110 PRINT A+B,(A+B)/2    ←—A,Bの合計と平均
```

105行は，プログラムに新しく付け加える行です．行番号の値に注目して下さい．105と言う数値は，100と110の間ですから，この行は100行と110行の間に追加されます．つまり，行と行の間に新しい行を追加したい時は，その2つの行の行番号の間の数値を新しい行番号にすればよいのです．実際に入力して，リストをとってみて下さい．

```
105 input b              |
110 print a+b,(a+b)/2    | 行の入力
list
100 INPUT A              |
105 INPUT B              | 変更されたプログラム
110 PRINT A+B,(A+B)/2    |
120 END                  |
Ok
run
? 2                      |
? 3                      | 実行させてみる
 5           2.5         |
Ok
■
```

このように簡単です．ただし，挿入はあくまで，目的の行番号と行番号の間の番号を選ばなければなりませんから，プログラムを入力する時は，ある程度の間隔をあけて行番号を付けておかなければなりません．通常のプログラミングでは，10番おきというのがポピュラーです．後で説明する行番号自動発生のコマンド（AUTO）も，初期状態では，10番ごとの行番号を発生してくれます．

次に不要になったプログラムを削除する方法をお話ししましょう．プログラムに変更を加えていて，ある行が不要になった時は，その行の行番号のみを入力すればよいのです．例えば105行を削除したい時は，105という行番号のみを入力し，リターンキーを押します．

```
105     ←—行番号のみを入力
list    ←—リストをとる
100 INPUT A              |
110 PRINT A+B,(A+B)/2    | 変更されたプログラム
120 END                  |
Ok
■
```

今の例では，1行を削除した訳ですが，複数行を一度に削除することもできます．その場合，

DELETE 命令を使います.

```
delete -110
Ok
list
120 END
Ok
■
```

この場合の範囲指定は，LIST 命令とほぼ同じですが，誤ってプログラムを削除してしまうことのないように，終点行番号を省略することは許されていません.

では，プログラム全部を削除したい時はどうすればよいでしょうか．当然 DELETE 命令でも可能ですが，もっとよい方法があります．それは NEW という命令を使えばよいのです．NEW 命令は現在あるプログラムをすべて削除する働きをします.

```
list
120 END
Ok
new    ←─NEWを入力
Ok
list   }リストをとっても何もない
Ok
■
```

NEW は新しいプログラムを入力する場合に，古いプログラムを全部抹消しておくために使います.

2.3.3 プログラムを保存する

皆さんが苦労して作ったプログラムを，保存しておくにはどうしたらよいでしょうか．プログラムは NEW を行ったり，PC-8801 の電源を切ったりすると消えてしまいます．また停電，もしくはプラグがコンセントから抜けてしまった，などの事故が生じるかもしれません．どちらにしても，プログラムが保存してあれば後でまた使うこともできますし，大変便利です.

PC-8801 は，プログラムを保存しておくための機能を幾つか持っています．それは次のようなものです.

① プログラムのリストを紙に印刷する.

② プログラムを外部記憶装置（フロッピーディスク，カセットテープ）に記録する.

①は，LIST 命令を使うとプログラムのリストがとれますが，それと全く同じものを紙に印刷する働きです．②は，現在記憶されているプログラムを，外部の記憶装置に記録する働きです．そして，その記録されたプログラムを PC-8801 に読み込むこともできます．②の機能があれば，プログラムの保存，そして再使用が自由にできます．①の機能は，プログラムを目に見える状態で保存するためのものです.

ところで，①のためにはプリンタが，②のためにはフロッピーディスクかカセットテレコが必要となります．

☆プログラムリストの印刷

プログラムのリストをプリンタを使って出力させたい時は，次の様に入力します．

LLIST ↲

プリンタが動き出し，プログラムのリストがプリンタ用紙に印刷されていきます．もしプリンタがつながっていないのに LLIST を入力すると，エラーメッセージが表示されます．

☆プログラムの外記記憶装置への記録，読み出し

操作法が，フロッピーディスクとカセットテープでは違いますから，別々に説明します．なお，プログラムの外部記憶装置への記録をセーブ，読み出しをロードと呼んでいます．

フロッピーディスク

プログラムのセーブには次の様に入力します．

SAVE〝ドライブ番号：ファイル名〟↲

ドライブ番号はディスケットの入っているドライブの番号で，ファイル名はプログラムに付ける名前のことです．例えば今あるプログラムに sample という名前を付けて，1番のドライブに入っているディスケットに記録したければ，次の様に入力すればよいのです．

SAVE〝1：sample〟↲

ドライブ番号は，ディスクユニットを御覧になれば分かりますが，1と2の数を使うことができます(ディスクユニットが一台の場合)．ファイル名はどんな文字でも使うことができます．英字の大文字と小文字も区別します．文字の数はいくらでもよいのですが，9文字までしか判別しません．それ以上の文字は無視されます．

プログラムをロードするには次の様に入力します．

LOAD〝ドライブ番号：ファイル名〟↲

ファイル名はセーブの時に使ったものと同じ名前を使用します．この指定されたファイル名のプログラムを見つけて，ロードします．たとえば，前の例でセーブした sample というプログラムをロードするには，次の様に入力すればよいのです．

LOAD〝1：sample〟↲

実際に，フロッピーディスクへのプログラムのセーブ，ロードを行うと，指定したドライブがカチッと音がしてランプが点燈します．しばらくして画面にOkが表示され，動作が終了します．もし指定されたドライブにディスケットが入っていないと，エラーメッセージが表示されます．またロードを行った時，指定されたファイル名のプログラムが見つからない場合もエラーメッセージが表示されます．

```
100 input a
110 print a,a*a,sqr(a)
120 end
list
100 INPUT A
110 PRINT A,A*A,SQR(A)
120 END
Ok
save "1:sample"
Ok
new
Ok
list
Ok
load "1:sample"
Ok
list
100 INPUT A
110 PRINT A,A*A,SQR(A)
120 END
Ok
■
```

カセットテープ

カセットテープの場合は，フロッピーディスクの方法と多少異なります．主な違いは，フロッピーディスクでのドライブ番号の指定が，カセットを指定するものに変わっている点です．

プログラムのセーブには次の様に入力します．

SAVE "CAS2：ファイル名" ⏎

ドライブ番号の代りにCAS2と入力することによって，カセットテープにプロブラムをセーブすることを指定しています．ファイル名については，フロッピーディスクでの説明と同じです．

実際にカセットテープにプログラムのセーブを行う時は，カセットテレコを録音状態にした後にリターンキーを押します．セーブが終了するとOkが表示されます．

プログラムをカセットテープからロードする場合は，次の様に入力します．

LOAD "CAS2：ファイル名" ⏎

これもセーブの場合と同じく，ドライブ番号の代りにCAS2を使います．

ロードを行わせると，カセットテレコより指定されたファイル名のプログラムを探します．もし途中で，別のファイル名のプログラムに出会った時は，次の様なメッセージを表示して，そのプログラムを読み飛ばします．

Skip：ファイル名

そして目的のファイル名のプログラムが見つかると，次の様なメッセージを表示し，そのプログラムのロードを開始します．

Found：ファイル名

目的のプログラムをロードしている間は，画面の右上に＊が点滅しています．ロードが正常に終了すればOkと表示され，何らかの異常（テープのキズや雑音）が生じたなら〝Tape read error（? TP Error）〟と表示されます．

カセットテープではセーブしたプログラムが正常かどうかを検査することができます．これはカセットテープの信頼性が，フロッピーディスクより低いことに対する配慮です．それには次の様に入力します．

LOAD？〝CAS2：ファイル名〟⏎

これは指定されたファイル名のプログラムと，PC-8801内部のプログラムを比較します．もし同一ならばOkと表示し，異なっていたならBadと表示します．なおファイル名で指定したプログラムの探査は，ロードの場合と同じように行います．

注)プログラムのロード，セーブの際のカセットの指定は，CAS2だけではなくCAS1も行うことができます．これはフロッピーディスクのドライブ番号の指定とは違って，2台のテレコを使える訳ではなく，テープへの記録のスピードが違っています．CAS2よりもCAS1と指定した方が、スピードは速くなっているのです．ちなみに，CAS1の方が1200ボー，CAS2が600ボーで，CAS1の方がCAS2より半分の時間でセーブ・ロードができます（ボーとはデータの転送速度の単位で，1秒間に何ビット送れるかを表わしたものです)．しかしスピードが速くなった反面，信頼性が低下してしまい，Tape read errorを生じる確率が高くなってしまいます．また，カセットテレコとの相性もシビアになってきます．ですから皆さんにはCAS2と指定してロード，セーブを行うことをお勧めしておきます．

本体

〝CAS1：～〟➡1200ボー

〝CAS2：～〟➡ 600ボー

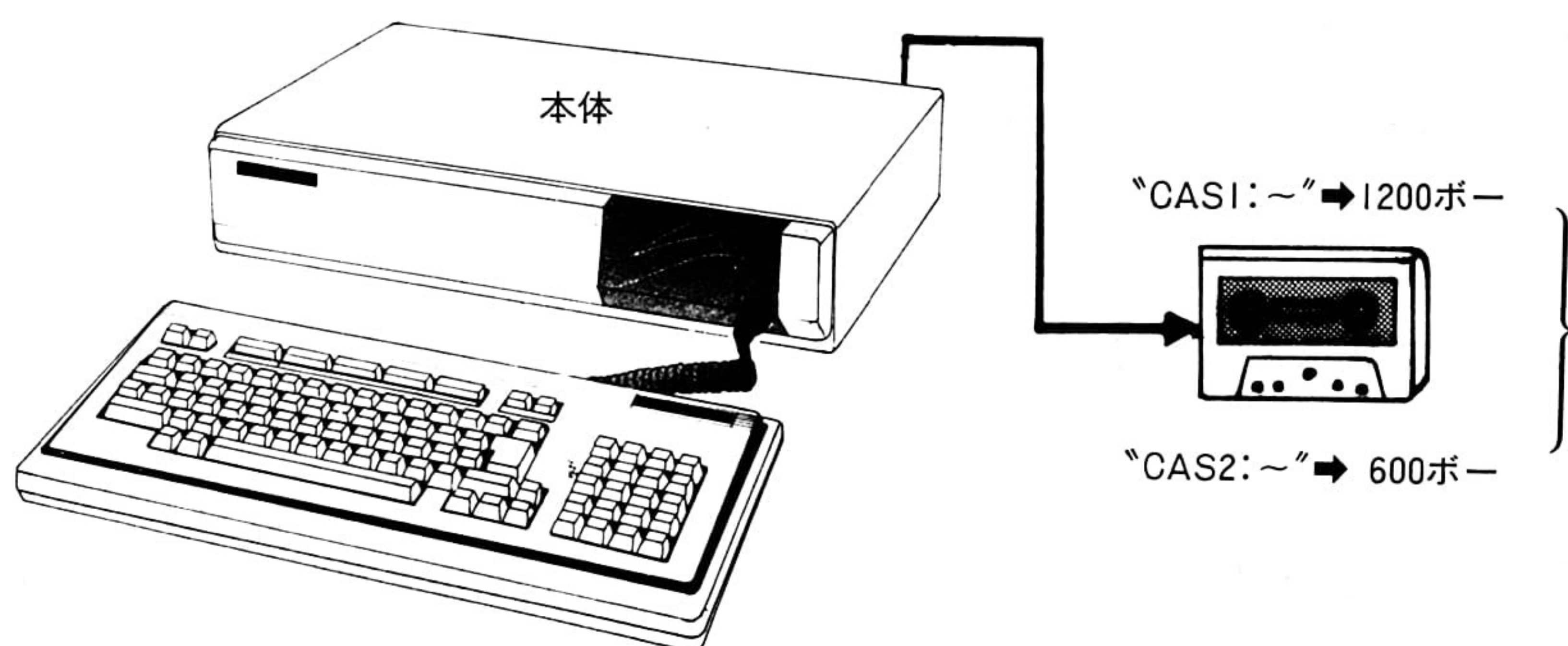



図2　2つの転送速度

2.3.4 プログラムの編集にはこのコマンドで

今までに，プログラムを編集するのに有用な幾つかの命令を見て来ましたが，ここでそれらの命令の機能をまとめておきましょう．また，まだ出てきていない便利な命令についても合わせて説明します．

ところで，LIST や SAVE などのプログラミングのための命令（プログラム全体を扱う命令）をコマンド，PRINT や INPUT などのプログラムの中で用いる命令をステートメントと呼ぶこともありますから，ちょっと言葉を覚えておいて下さい．

LIST

プログラムのリストを表示します．

特定の行を表示する機能もあります．

①LIST　100

100行のみを表示します．

②LIST　110-140

110行から140行までを表示します．

③LIST　130－

130行より後の行をすべて表示します．

④LIST　－170

始めより170行までの行を表示します．

RUN

プログラムを実行させます．

指定した行からプログラムを実行させることもできます．

RUN　130

130行からプログラムを実行させます．

NEW

現在あるプログラムをすべて抹消します．また変数の内容もすべてクリアします．

DELETE

特定の行のみを削除します．行番号の指定は LISTと同じように行います．但し，終点行番号の省略はできません．

DELETE　120－150

120行から150行までを削除します．

AUTO

行番号を自動的に発生させます.

AUTO　開始行番号，行番号間隔

AUTO　100，20

100行から20おきに行番号を発生させます.

RENUM

プログラムの行番号を付け直します．これを使うと，追加，削除などを行って不揃いになった行番号の間隔を，一定にすることができます.

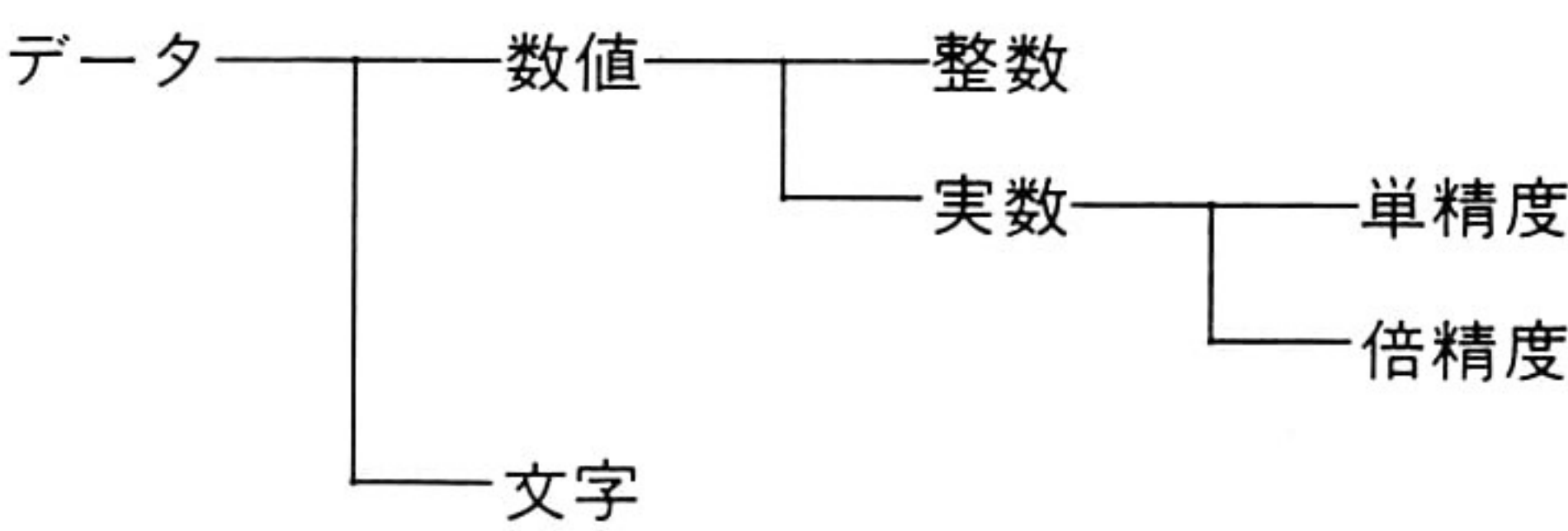
RENUM　100

プログラムの行番号を新しく100行から10おきに付け直します.

2.4 N88-BASICで扱えるデータ

今までに BASIC で扱って来たデータは，数でした．この数を私たちは，何の説明も無しに使って来ました．しかし，私たちが扱う数には，実数や整数などいろいろな種類があると同じように，BASIC にも扱う数には幾つかの種類があります．また扱えるデータは数だけではありません．文字も同じように扱えるのです.

ここではこれらのデータの種類について説明していきます．まず次の表を見て下さい．これは N88-BASIC の扱えるデータの種類を示しています.

```
データ──┬──数値──┬──整数
        │        └──実数──┬──単精度
        │                  └──倍精度
        └──文字
```

数値の種類で整数と実数は皆さんもよく知っておられるでしょう．しかし単精度，倍精度などは聞き慣れない言葉だと思います．これは有効数字の桁数が違うもので，コンピュータ内部の処理上の都合からこうなっています．整数と実数の区別も処理上の違いから生じたものです．人間は数値を扱うのに整数，実数などの区別なく，全部一緒に頭の中で処理することができますが，コンピュータはそうはいきません．きちんと区別して処理しなければなりません.

このようにデータの種類を分けたものを型と呼んでいます．整数は整数型，実数は実数型と呼びます．実数型の中に単精度型と倍精度型の区別がある訳です．このようにデータを型に区別すると，コンピュータが能率よくデータを扱うことができるようになります．ですからデータを扱う時は，そのデータの内容により一番適した型を使うようにしましょう.

整数型	□□	(2 byte長)
単精度型	□□□□	(4 byte長)
倍精度型	□□□□□□□□	(8 byte長)
文字型	□------□	(任意長，255文字（byte）以内)

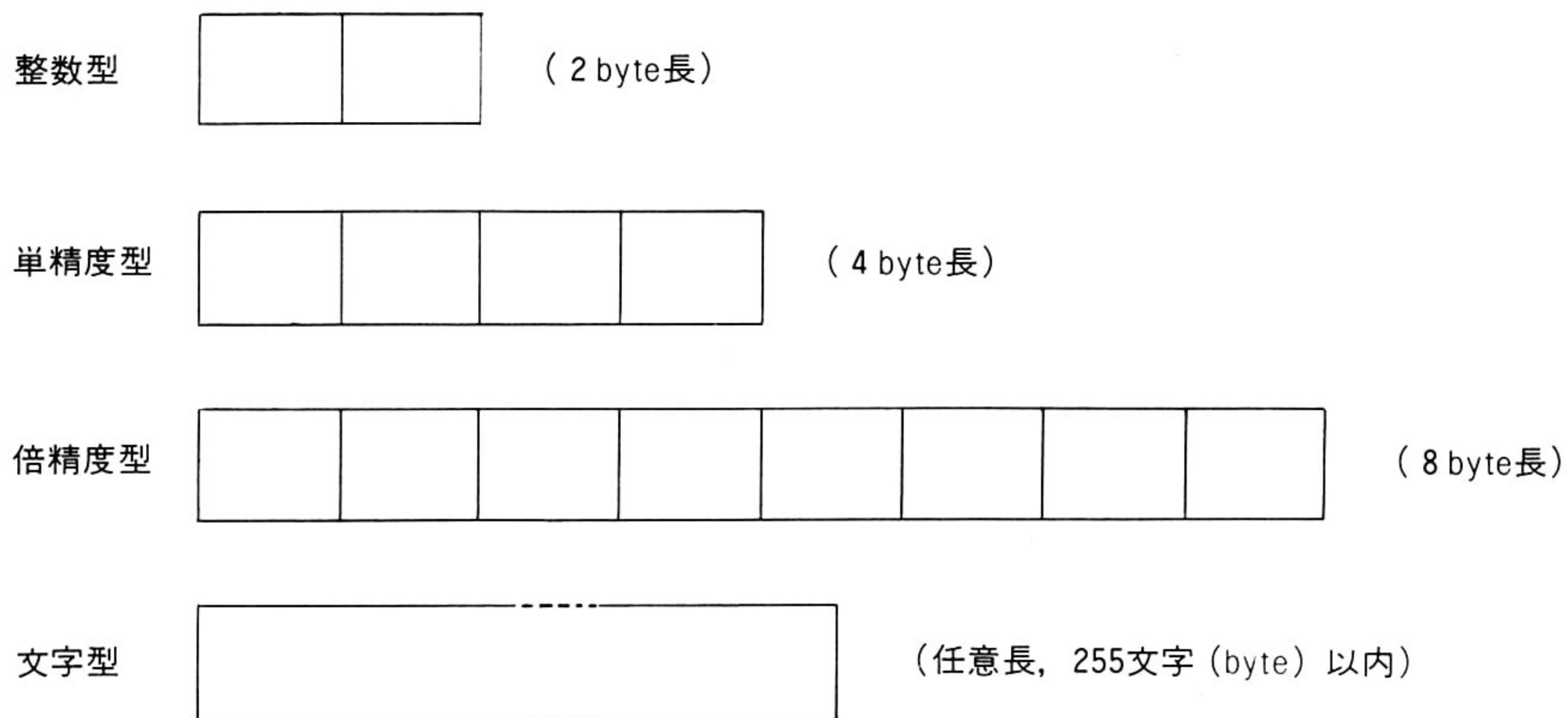



図3　データとbyte長

2.4.1　整数型は範囲が問題

整数型の数値は－32768から＋32767までの整数です．この数字を初めて見る人にとっては奇妙な数かもしれませんが，図のように2バイトで表すことのできる最大と最小の数であり，コンピュータの世界ではおなじみの数字です．また，たったこれだけと思われるかもしれませんが，これだけの値に限定することによって，大きな利点が生じます．それはコンピュータの処理速度が向上することです．さらに，整数型の変数は実数の変数に比べて，記憶領域（メモリ）の消費が半分以下になるのです（変数の型については後で出て来ます）．

この整数型の数値を用いるには，－32768から＋32767までの数を使うか，または数の後に％（パーセント記号）を付ければよいのです．この％はその数を実数型などと区別して，特に整数型と指定する場合に用います．

```
print 1.3%,2.6%,-3.5%
 1             3             -4
Ok
■
```

注）整数型の数値には他にも8進形式と16進形式があります．

10進形式	8進形式	16進形式
123	&173	&H7B
	&O173	

2.4.2　実数型には2つの精度

実数型の数値は単精度型と倍精度型の2つがあります．また数値の表現形式として，固定小数点形式と浮動小数点形式があります．浮動小数点形式の指数部の数値の範囲は－39から＋38までです．

固定小数点形式

　324.57

浮動小数点形式（いわゆる指数形式です）

　3.2457×10^2

　3.2457−E2　（単精度型）

　3.2457−D2　（倍精度型）

2.4.3　単精度型は7桁以内で

単精度型の数値は，有効数字が7桁以下の固定小数点形式か，または浮動小数点形式で表された実数です．

BASICが数値を扱う時は，単精度型として扱うことが暗黙の了解となっています(変数の型も同じです)．もし，特に単精度型ということを表したい場合は，Eを使った指数表現(1.2E3など，このEは必ず大文字でなければならない）を用いるか，数値の後に！（イクスクラメーション記号）を付けたものを用いればよいのです．なお単精度型の有効数字の桁数は7桁ですが，数値を表示させた時には6桁に丸められます．

```
print 1.234567,0.000001234,1.234E-3
 1.23457         1.234E-06      .001234
Ok
print 123456789!,1.234567E6,1.234567E-6
 1.23457E+08    1.23457E+06    1.23457E-06
Ok
■
```

2.4.4　倍精度型の精度は倍以上

倍精度型の数値は，有効数字が16桁までの固定小数点形式か浮動小数点形式の実数です．

倍精度型は単精度型と比べて，2倍以上の有効数字の桁数を持つ数値を表現できます．そのため桁数の大きい整数を扱う時や，より精度の高い計算を行う場合などに用いることができる訳です．

この倍精度型の数値を表すには，8桁以上の数値か，もしくはDを使った指数表現(1.23D3など，Dは必ず大文字を用いる）を使えばよいのです．もし，7桁以下の数値で倍精度型を使いたい場合は，数値の後に#記号を付けて倍精度型ということを表します．

```
print 123456789,0.000012345678
 123456789      .000012345678
Ok
print 1/3,1/3#
 .333333        .3333333333333333
Ok
print 12345678901234567,1.23D-13
 1.234567890123457D+16       .000000000000123
Ok
■
```

2.4.5 文字型は255文字の範囲で

BASICでは文字を数値と同じように取り扱うことができます．文字列を定数として扱ったり，変数に文字列を代入したりすることができるのです．また文字列を結合したりする演算を行うこともできるのです．これらのことは，扱うデータの型を数値型から文字型に変えるだけで，同じ手続きで行うことができます．

文字型において数値定数に対応するものは，文字定数です．文字定数とは引用符（"）で囲まれた文字の列です．次の様なものが文字定数です．

"This is a pen."

"2001"

"カタカナ"

文字定数は数値定数と同じように，PRINT 文で表示させることができます．

```
print "This is a pen."
This is a pen.
Ok
■
```

また四則演算の＋を用いて，2つの文字定数を結合させることもできます．

```
print "PC-"+"8801"
PC-8801
Ok
■
```

2.5 変数の名前(変数名)と型

BASICを電卓風に使った時に述べたように，変数とは，値を自由に決めたり，取り出したりすることのできる入れ物です．この入れ物に付けた名前が変数名で，これを使って変数の値を参照したり，別の値を代入することができます．

もっと正確な言い方をすれば次の様になるでしょう．BASICが取り扱うデータ（数値や文字列）を格納しておくための内部の記憶領域に，変数名という名前を対応させて，その名前によってデータを扱えるようにしたものが変数です．

2.5.1 区別は変数名で

変数は変数名によって区別されて取り扱われます．この変数名にはいろいろな名前を用いることができますが，次の様な規則を守らなければなりません．

① 英字で始まること

② 文字の長さは最大40文字であること

③ 英数字とピリオド(.)以外は使用しないこと

④ 名前が予約語そのものでないこと

予約語というのはBASICで使われる命令に付けられた名前（RUN, PRINTなど）のことです．つまり，RUNとかPRINTという変数名は許されない訳です．またFNで始まる名前も変数名には使えないことになっています．なお，英字の大文字と小文字は区別されません．

変数名の例を幾つか示します．

A, ABC, DC10, PRINTX

変数名にならないもの，

1A, RUN, ヘンスウ, J&B

2.5.2 4種類の型

BASICが扱うデータにはいろいろな型がありましたが，それぞれの型のデータを変数によって取り扱いたい場合には，変数にも同じ型のものを用いなければなりません．整数型の数値には整数型の変数が必要となるのです．

変数の型としては，前に述べたデータの型と同じ整数型，単精度型，倍精度型，文字型の4つがあります．変数がどの型であるかを指定するには，変数名の後に型を表す記号を付ければよいのです．この記号は型宣言文字と呼ばれるもので，その変数の型を宣言します．それらは次の様に決められています．

型宣言文字		
	%	整数型
	!	単精度型
	#	倍精度型
	$	文字型

これらの型宣言文字を見ると，文字型を除いて，数値の型を表す記号と同じものであることが分かるでしょう．数値の型と変数の型を表す記号は同じものが使われているのです．

ところで，今まで使って来たAなどという型宣言文字の付いていない変数は，どの型となっているのでしょうか．それは単精度型なのです．変数を型宣言文字を付けないで用いると，この変数には！が付いているものとみなされて単精度型とみなされます．

A%　整数型変数

A
A!　} 単精度型変数

A#　倍精度型変数

A$　文字型変数

同じ変数名でも型宣言文字が違うと，別の変数として取り扱われます．つまり同じ変数名でも型が違えば区別して使うことができるのです．

```
100 a=3.2109
110 a%=342
120 a#=1.234567890123
130 print a,a%,a#
140 end
run
 3.2109         342             1.234567890123
Ok
■
```

注）変数の型を宣言する方法がもう1つあります．それはDEFINT, DEFSNG, DEFDBL, DEFSTRの型宣言文を使う方法です．これらは3章で詳しく説明します．

2.5.3　型を合わせる

BASICでデータを取り扱う場合は，そのデータの型をはっきり決めておかねばなりません．そしてその型で扱える範囲内でデータを処理しなければならないのです．もし，データの値が，その範囲を越えるとエラーが発生します．

```
print 40000%
Overflow        ←―整数型の値の範囲をこえた
Ok
print 1.23E40
Overflow
 1.70141E+38
Ok
print 1.23D40
Overflow
 1.701411834604693D+38
Ok
■
```

何も型宣言などせずに数値を扱っても，それは単精度型のデータだと暗黙のうちに決められているのです．

BASICはオーバーフロー（桁あふれ）が生じると，データの正しい値をとることができなくなります．ですから，ある型を用いてデータを取り扱う時は，その型で扱える値の範囲を，よく把握しておく必要があります．

ではいろいろな型のデータを混ぜて処理を行うとどうなるでしょうか．例えば，ある型の変数に他の型のデータを代入したり，計算式にいろいろな型が混在している場合などです．こんな場合には，BASICは必要に応じてデータの型を変換して処理を行ってくれます．ただし，変換が行われるのは数値型のデータのみで，文字型と数値型との間では変換は行われません．

まず，変数への代入において，この働きを見てみましょう．ある型の数値が他の型の変数に代入されると，その数値は，変数の型に合うよう変換されて，代入が行われます．

```
a%=3.14
Ok
print a%
 3
Ok
■
```

実数が整数に変換される場合は，小数点以下が四捨五入されます．もし整数に変換したものが，整数型の範囲を越えた場合はエラーとなります．

```
a%=1.23E5
Overflow
Ok
■
```

倍精度型の実数が単精度型の実数に変換される場合は，有効数字を7桁に丸めたものとなります．

```
new
Ok
100 a#=1.23456789D10
110 a=a#
120 print a#,a
130 end
run
 12345678900    1.23457E+10
Ok
■
```

（単精度型の数値の表示は6桁で行われることを思い出して下さい）．

逆に単精度型が倍精度型に変換される場合は，ちょっと問題が生じます．それは有効数字の桁数の問題です．つまり，単精度型を倍精度型に代入しても，有効数字の桁数は単精度型のままなのです．

```
100 a=1.234567
110 a#=a
120 print a#,a
130 end
run
 1.234567046165466               1.23457
Ok
■
```

ですから，倍精度型の変数に有効数字が7桁以下の数値を代入する場合は，数値の後に#を付けて倍精度型の数値であることをはっきり示す必要があります．

```
100 a#=1.234567
110 b#=1.234567#
120 print a#,b#
130 end
run
 1.234567046165466             1.234567
Ok
■
```

一方，整数型が倍精度型に変換される場合は，小数点以下の誤差は生じません．

```
a#=123%
Ok
print a#,a#-123#
 123            0
Ok
■
```

今度は，型の違う数値の間での演算を見てみましょう．型の違う数値間の演算は，精度の高い方の型に変換されて演算が行われます．

```
100 print 1000#/3#
110 print 1000#/3!
120 print 1000!/3#
130 print 1000!/3!
140 end
run
 333.3333333333333
 333.3333333333333
 333.3333333333333
 333.333
Ok
■
```

異なる型の数値を取り扱う場合は，これらのことに十分注意して行って下さい．

2.6 式を見る

式とは，定数や変数を演算子で結合したものです．このような一般的な数式の他にも，定数や変数だけのものも式と呼びます．ちょっと言葉では表しにくいものですが，今までに出て来た式としては，代入文の右辺やPRINT文で値を表示させたものなどがあげられます．式の例をいくつか示します．

A＋2	変数と定数の加算
3.14	数値定数のみ
“PC-8801”	文字定数のみ
2＊SQR（2）	定数と関数の乗算

四則計算を行わせる計算式は，まさしく式という感じがします．しかし定数や変数が1つだけのものも式と呼ぶことに注意してください．文字定数のみの場合も式と呼びます．

ここで，PRINT 命令の働きを思い出してください．PRINT は，計算式や変数の値を表示します．つまり式の値を表示する働きを PRINT はするという訳です．ですから，PRINT に続くものが式であれば，数値でも文字でも表示してくれます．

```
print 3.14159
 3.14159
Ok
print "Open the door."
Open the door.
Ok
■
```

2.6.1 演算子いろいろ

演算子とは演算を行わせる記号のことです．今までに出て来たものとしては，四則演算を行わせる＋,－，＊,／があります．これらは数値演算を行わせることから，数値演算子と呼ばれています．

演算には数値演算の外にもいろいろなものが BASIC で行うことができるようになっています．どの演算もプログラムを作る上で，大変重要なものばかりです．ところで演算子は演算を行わせるもので，定数や変数を演算子で結んだものが式でした．こう考えると，式というのは演算の内容を記述したものと言うことができるかもしれません．

☆数値演算子

数値演算子には今までに出て来た＋(加算)，－(減算),＊(乗算)，／(除算)，－(負号)のほかにも，∧(累乗)，￥(整数の除算)，MOD(剰余の演算) というものがあります．

累乗というのは，同じ数をかけ合わせる演算で，２の３乗とかAのB乗などの計算を行うことができます．

使い方	意味
2∧3	$2^3=2\times2\times2$
A∧B	A^B

```
print 2^3, 2^0.2, 2^-2
 8             1.1487         .25
Ok
print 2^2^2, -2^2
 16            -4
Ok
■
```

計算例をよくみて下さい．2∧2∧2は $(2^2)^2$ となり，－2∧2は$-(2^2)$ となっています．累

乗は左から順に計算され，負号よりも先に行われることが分かると思います．

¥と MOD は数値演算子の仲間ですが，一般の数式には使われることはありません．BASIC 独特のものです．

¥は，結果が必ず整数になる除算を行う演算子です．割る数，割られる数とも整数に丸められて，結果も整数（小数点以下切り捨て）で求められます．

```
print 9¥5, 9.4¥4.6, 4¥5
 1             1             0
Ok
■
```

MOD は整数の除算を行った後の余りを求めるものです．

```
print 9 mod 5, 5 mod 6, -6 mod 4
 4             5            -2
Ok
■
```

ところで，これらの演算子の演算順序はどうなっているのでしょうか．累乗は負号よりも早くなっていました．¥と MOD は乗除算より遅く，加減算よりも早くなっています．つまり演算順序は，カッコの中，累乗，負号，乗除算，¥と MOD，加減算の順になります．もし累乗などがあって演算の順序が分かりにくければ，積極的にカッコを用いて分かりやすく式を書けばよいのです．

☆関係演算子

関係演算子は，2つの値を比較するためのものです．この演算子を使って2つの値の大小関係や，2つが等しいかどうか，などを調べることができます．演算子は，=，<，>の3つから構成されており，それぞれを単体で用いたり，2つを組み合わせたりして用います．数学での等号，不等号といった言葉を思い出して下さい． この演算子で表す等号や不等号が， 2つの値の関係として正しいか間違っているかを調べてくれます．そして正しければ真として（−1）を，間違っていれば偽として（0）を結果とします．

3つの演算子を組み合わせると，合わせて4通りの比較を行うことができます．

● =

2つの値が等しいかどうかを調べます．

```
print 2=2, 2=3, 2.2=2.21
-1             0             0
Ok
■
```

最初の 2 = 2 は正しいですから −1(真)の結果が，2 番目は 2 = 3 で正しくないですから 0 (偽)

の結果が得られたのです．

● <, >

2つの値の大小関係を調べます．

```
print 2<3, 2>3, 2<2
-1            0             0
Ok
■
```

1番目は真，2番目は偽，3番目も偽の結果が得られています．

● <=, =<

どちらも同じ意味で，左辺の値が右辺の値より小さいか，または等しいかを調べます．

```
print 2<=3; 3=<2, 2=<2
-1            0            -1
Ok
■
```

● >=, =>

どちらも同じ意味で，左辺の値が右辺の値より大きいか，または等しいかを調べます．

● <>, ><

これらは，2つの値が異なっているかを調べます．

```
print 2<>3, 3><2, 2<>2
-1           -1             0
Ok
■
```

関係演算子として＝（等号）が使われていますが，これは代入文の等号とは全く違うものです．混同しないようにしましょう．関係演算子の＝は演算子であり，式の中で用いられます．一方，代入文の＝は変数と式を結んで，式の値を変数に代入することを示しているのです．ちょっと次の例を見て下さい．何をやっているのか分かりますか．

```
a=2
Ok
b=a=a
Ok
print b
-1
Ok
■
```

2番目に実行しているのは代入文です．変数Bに A＝A という式の値を代入しているのです．A＝A の結果は真ですから B には－1が代入されています．同じ＝でも意味が全く違うことに注意して下さい．

この関係演算子と数値演算子を組み合わせて式を作ることができます.

```
print 2+3=5, 2+(3=5)
-1            2
Ok
■
```

関係演算子は数値演算子よりも後に演算が行われることが，この例から分かるでしょう.

☆論理演算子

論理演算子は,主に関係演算子を使った式などと組み合わせて,複数の値の関係を調べるために用います. 関係演算子での演算の結果として得られた，真（−1）と偽（0）の2つの値を操作して，真と偽のどちらかの値を結果として与えます. これを論理演算と呼びます.

論理演算子の代表的なものは，AND，OR，NOTです.

AND　論理積

2つの値（真と偽）の論理積をとります.

A	B	A AND B
真	真	真
真	偽	偽
偽	真	偽
偽	偽	偽

2つの値がどちらも真のときのみ，結果が真となります.

OR　論理和

2つの値の論理和をとります.

A	B	A OR B
真	真	真
真	偽	真
偽	真	真
偽	偽	偽

2つの値のどちらか一方あるいは両方が真のときに，結果が真となります.

NOT　否定

NOTは1つの値しか操作しません. 値の否定をとります.

A	NOT A
真	偽
偽	真

値の真偽を反転させます．

実際にこれらの論理演算子と関係演算子を組み合わせて式を作ってみます．

```
print 1<a and a=<5, a<-2 or 1<a, not a=3
-1               -1              0
Ok
■
```

但し，aは3とする．

最初の式はANDの左辺が真，右辺も真ですから結果は真となります．2番目はORの左辺が偽ですが，右辺は真ですので結果は真となります．3番目はNOTによって真が偽となっています．ところで3番目の式はNOTを使わなくても，A<>3と書けばよいことが分かりますか．

ここまでで，ANDとORをどんな場合に使えばよいか分かりましたか．ANDは両辺の2つの関係がどちらも成立した場合（どちらも真）に真となります．一方ORは，2つの関係のどちらかが成立した場合に真となるのです．2つの働きの違いをよく摑んでください．

今まで，関係演算子と論理演算子を使っていろいろな式を作って来ましたが，これらの式は，すべて結果の値が真（−1）か偽（0）のどちらかにしかなりませんでした．このような，結果が真か偽のどちらかになる式を，論理式と呼んでいます．論理式は今までのことから分かるようにいろいろな関係が成立したかどうかを調べる条件判断のために使うことができます．この論理式は後で出てくる，条件判断を行う命令の中で使うことが多いのです．

さて数値演算子が，一通り出揃った所で，各演算子の演算順位をまとめておきましょう．最後に出て来た論理演算子は演算順位が最も低くなっています．

演算子の演算順位

① カッコでくくられた式

② 数値演算子

③ 関係演算子

④ 論理演算子（NOT，AND，ORの順です）

注） 論理演算子は説明したものの外にもいくつか用意されていますが，あまり使わないのでここでは省略しました．また論理演算において実際にコンピュータ内で行われている処理はビット演算なのです．

☆文字列に演算を施す

BASICでは数値型のほかに文字型のデータを扱うことができますが，文字型のデータに対していくつかの演算を行うこともできるのです．その演算は，文字列の結合と比較です．

文字列の結合

文字定数や文字変数を，演算子＋でつなぐことによって１つの文字列にすることができます．言うなれば文字列のたし算です．

```
new
Ok
100 a$="This is a "
110 b$=a$+"book."
120 print b$
130 end
run
This is a book.
Ok
■
```

文字列の比較

数値の比較に用いた関係演算子を使って，文字列を比較することができます．比較は両方の文字列の先頭から１文字ずつ，そのキャラクタコード（キャラクタを順序だって並べて，それに頭から番号をふったもの，詳しくは後述)が比べられていきます．もしどちらも同じ文字列ならば，その２つは等しい訳です．もし途中で文字が違った場合は，その文字のキャラクタコードの大きい方の文字列が大きくなります．また途中でどちらかの文字列が終ってしまった場合は，長い方の文字列が大きくなります．

```
print "a"="a", "a"="A", "a">"A"
-1             0            -1
Ok
print "abc"="abc", "ab">"aB", "ab"<"abc"
-1             -1           -1
Ok
■
```

文字列の比較は，文字変数の内容を調べたり，文字列のソートに使うことができます．

2.7 代入文を考える

代入文は今までにたくさん使って来ました．皆さんも大分使い慣れて来たのではないでしょうか．ここで，もう一度，代入文について注意すべき事柄を説明しておくことにします．

代入文は次の様な書式で用います．

変数名＝式

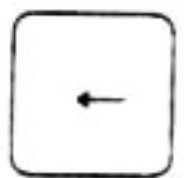
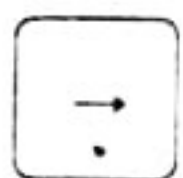
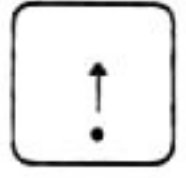
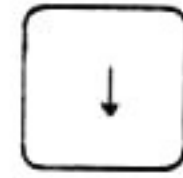
変数名は式の値を代入する変数を示しています．式で表わされる値はどんな型のものでも構いません．ただし，変数と式の値の型は，数値型か文字型のどちらかに統一されていなければなりません．数値型に文字型を代入するようなことはできません．その逆ももちろんです．数値型での代入においては，異なる型の変数への代入を行うことができます．この場合は，式の値の型が

変数の型に変換されて代入が行われます.

例　A＄＝"PC-8801"　　**文字型の代入文**

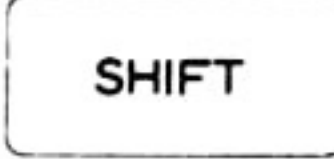
　　A＝A＋2　　　　　**数値型の代入文**

変数に何も値を代入しないで用いた場合，その値は，数値変数では0，文字変数ではヌルストリングとなっています．ヌルストリングとは，長さが0の文字列で，文字変数に何も入っていない状態を表しています．文字定数として""（ダブルコーテーション）を用いれば，ヌルストリングを変数に代入したりすることができます．

2.8 便利なスクリーン・エディタ

今までのプログラムの編集を行う方法は，行単位で，修正や挿入，削除を行う方法でした．しかし，この方法だと，ある行の一部分を修正したい場合でも，その行全部を新しく入力しなおさなければなりません．この N_{88}-BASICには，ディスプレイを活用してプログラムの修正などが簡単に行えるように,スクリーンエディタと呼ばれる画面を使った編集機能が備えられています．この機能を用いると，修正する行を全部入れ直すこともなく，大変能率よくプログラムの編集作業を行うことができるようになります．

2.8.1 必ず押そうリターンキー

今まで，直接モードで命令を入力して実行させたり，プログラムを入力したりする時は，必ずリターンキーを押すことを約束してきました．このリターンキーを押すことは，命令やプログラムを入力したことをBASICに知らせる合図だった訳です．これから説明するスクリーンエディタにおいても,プログラムの修正などが終わったならば,その行ごとにリターンキーを押して，新しくなったプログラムの入力の合図を送らなければなりません.くれぐれも各行の修正が終わるごとにリターンキーを押すことを忘れないで下さい．もしリターンキーが入力されないと，いくら編集作業を行っても，全く変更はされません．

2.8.2 カーソルで自由自在

カーソルはキーボードから，入力される文字がどこに表示されるかを示しています．つまり，カーソルは，文字の表示される位置を常に示している目印というわけです．

このカーソルをプログラムの行の書き換えたい部分まで移動させて,新しい文字を入力すれば,その文字が,もとあった文字と置き換わります.これを行うためにカーソルを動かすには，[←][→][↑][↓]のカーソル移動キーを使います．

2.8.3 文字を消去する

プログラム中に消したい文字がある場合は，カーソルをその文字の次の文字に移動させ，[INS DEL]のデリートキーを押します．カーソルから右の文字列が，１文字分左へ移動します．

2.8.4 文字を挿入する

プログラムの中に何文字か挿入したい場合は，挿入したい場所の次の文字にカーソルを移動させ，[SHIFT]＋[INS DEL]のインサートキーを押します．こうすると，入力がインサートモードとなり，次から入力する文字はカーソルの前に挿入され，カーソルから右の文字列が，１つずつ右に移動していきます．インサートを終わらせるには，カーソルを移動させるかリターンキーを入力します．

例 100 inprint B$ を 100 input ASB$ に修正します．

カーソルを inprint の r の文字に移動させ，u の文字を入力します．

100 inpu i nt B$

カーソルを t まで移動させ，デリートキーを２回押します．

100 inpu t B$

カーソルを B まで移動させ，インサートキーを押します．

100 input B

A と S の文字を入力します．

100 input AS B $

修正が終ったので，最後にリターンキーを押します．

スクリーンエディタの基本的な操作は，カーソルを自由に画面上に動かして，画面上すでに表示されている命令やプログラムを修正して再び入力することです．次の操作を行ってみて下さい．

①PRINT 2*3と入力する．

⇨ 結果が６と表示されます．

②次にカーソルを動かして，２＊３を３＊３に変更する．

③リターンキーを押す．

⇨ ９という結果が表示されましたね．

ここで，リターンキーを押す意味を思い出して下さい．リターンキーは，入力の合図でした．このリターンキーが押されることに対して，BASIC はどんな動作をしているかといいますと，リターンキーが押されると BASIC は，カーソルが置かれている行の文字列（命令やプログラム）が，全部入力されたものとみなしているのです．つまり，リターンキーは，画面上で表示されているカーソルの置かれた行の入力の合図という訳です．ですから画面上に何か表示してあれば，カーソルをその行に移動させてリターンキーを押すだけで，それが全部入力できるのです．これがスクリーンエディットの意味なのです．

2.8.5 画面を消去する

スクリーンエディタは画面上で文字の編集を行いますから，不要な文字が表示されていたりすると，都合の悪いことになります．画面上に表示されているものをすべて消去するには，HOME CLR のクリアキーを押します．画面が全部クリアされて，カーソルは画面の左上隅に移動します（画面の左上隅の位置をホームポジションと呼んでいます）．

同じキーを使って，画面を消さないでカーソルをホームポジションに移動させることもできます．それには，SHIFT ＋ HOME CLR のホームキーを押します．これはカーソル移動のためのキーです．

2.8.6 EDITコマンドを使う

EDIT コマンドは，スクリーンエディタの機能を使って，プログラムを修正するのに便利なコマンドです．使い方は，EDIT 行番号と入力します．

こう入力すると，画面がクリアされて行番号で指定した行が表示され，その行の先頭にカーソルが置かれます．そしてスクリーンエディタを使って，その行を修正すればよいのです．

N_{88}-DISK BASIC の EDIT コマンドでは，ROLL UP，ROLL DOWN のキーを使って指定した行の前後のプログラムも画面に表示させて修正を行うことができます．このキーを押すとプログラムをまるで巻き物を見るように，自由に表示させることができるのです．

2.8.7 エラー箇所の手直し

プログラム，あるいは直接モードでの命令の実行中にエラーが発生した場合は，その実行を中断します．この時 BASIC は，エラーメッセージとエラーの生じた行の行番号を表示します（直接モードでは行番号は表示されません）．

プログラム中でエラーが生じた場合に，その行が表示され，エラーのある項目か，またはその次の項目の先頭にカーソルが置かれることがあります．この時はすぐにエラーのある所の訂正ができますから，大変便利です．この表示は Syntax error の場合には必ず行われるようです．

```
new
Ok
100 primt 1/3
110 i="abc"
120 end
run
Syntax error in 100
Ok
                    ↙カーソルが表示される
100 PRIMT 1/3
        ↑
        間違っている所
```

しかしすべてのエラーに対して，エラーの生じた行の表示を行ってくれる訳ではありません．エラー行の表示が行われなかった場合は，HELP のヘルプキーを押すことによって，前と同様な表示を行わせることもできます．

```
100 PRInT 1/3
run
 .333333
Type mismatch in 110
Ok                  ←―エラーの行の表示は行われない

110 I="abc"■  ←―HELPキーを押すと表示される
```

このヘルプキーは，今までのパソコンには無かったもので，大変便利な機能です．直接モードでの命令の実行においても，エラーの訂正が，いちいちカーソルを元に戻さなくても簡単に行える訳です．

エラーの生じた行の前後のプログラムを見たい場合には，前に述べたEDITコマンドを使うと便利です（N_{88}-DISK BASICの場合）．このときの行番号の指定に便利な方法があります．それは行番号の代わりに，．（ピリオド）を使うのです．これはBASICが現在着目している行の行番号を与える働きをします．BASICはプログラムを扱う時は，常に行番号を参照しながら行っています．ですからプログラムの実行や表示が中断した時に，ピリオドを使えばBASICが最後に参照した行の行番号が得られる訳です．この働きは，行番号を扱うすべてのコマンドで使うことができます（2.3　プログラムの編集を参照）．

2.8.8　便利なコントロール機能

N_{88}-BASICには，スクリーンエディットの手助けとなる，便利なコントロール機能が幾つか備えられています．これらの機能は，コントロールキーと英字のキーを一緒に押すことによって実行できます．なお，これらの機能を使ってプログラムを修正した場合も，終わりにリターンキーを押すことは，前と変りありません．

☆CTRL-B

カーソルを項目（命令，変数，定数）ごとに，左へ移動させます．

☆CTRL-F

CTRL-Bとは逆に，項目ごとに右へカーソルを移動させます．

☆CTRL-X

カーソルをその行の一番後ろに移動させます．

以上の3つの機能は，カーソルを素早く移動させるためのものです．

☆CTRL-E

カーソルのある位置から，プログラムの行の終りまでを消去します．

☆CTRL-U

カーソルのある行の先頭から最後までを消去します．

☆CTRL-D

カーソルのある項目を，カーソルの位置からその項目の終りまでを消去します．

以上の3つは，文字の消去のためのものです．

☆CTRL-J

インサートモードで使うことによって，1行を2行に分割します．インサートキーを押してインサートモードにした後，CTRL-Jを入力すると，その行のカーソルから右の部分が次の行に移動します．移動した行の先頭に行番号を付ければ，1行を2行に分けて書いたことになるのです．この機能は，マルチステートメント（後述）で書かれた行を分割する場合などに使用します．

以上のようなコントロール機能をスクリーンエディットにおいて活用すれば，プログラムの修正をスピーディーに，且つ効率的に行うことができます．

3章 プログラミングの基礎

3.1 プログラミングに絶対必要な基本ステートメント

今までにBASICとは何であるかを説明するために幾つかの命令を使って来ましたが，ここからはプログラムを作るために必要なBASICの命令について本格的に説明していきます．

ところで皆さんは，BASICがどういう目的で考え出された言語であるか御存知ですか．最初は初心者に理解し易く，プログラミングの容易な言語として作られたのです．最近のパソコンの発達に伴って，BASICもどんどん進歩し，より多くの機能を備えたものが使われるようになって来ました．しかし機能が増えると共に，命令の数は増え，使い方は複雑になって来たことも確かです．ですが全部の機能を使わなければプログラムが作れない訳ではありません．そこで，ここでは使う命令の数をしぼって，その使い方とプログラムの作り方についての基礎を皆さんにお話しすることにしました．ここで選んだ命令はBASICの基本的なものですが，これさえ理解していれば，大抵のプログラムを作るには不自由しないというものです．

3.1.1 5つの基本ステートメント

ここで説明する命令は次の5つです．

①PRINT　②INPUT　③GOTO

④IF～THEN～ELSE　⑤END

そして，プログラムを作るために理解して頂きたいことは次の5つです．

①（数値や文字の）出力

②（数値や文字の）入力

③プログラムの分岐

④条件判断

⑤プログラムの終了

これらは，前の5つの命令の働きにそれぞれ対応しています．もちろん，変数や式，代入文な

どについても理解していなければなりません．ではそれぞれの命令が，どんな働きをするのかをお話ししましょう．

☆PRINT（結果の出力）

PRINT は今までにたくさん使って来ました．ここでもう一度，その働きを確認する意味で，説明をしていきます．PRINT の働きは，その後に続く式の値を画面に表示（出力）するものです．つまり PRINT の後に，定数や変数，式などを書けば，その値を求めて画面に表示してくれる訳です．

```
new
Ok
100 print 3
110 print "ascii"
120 print 2/3
130 end
run
 3
ascii
 .666667
Ok
■
```

また式は1つだけでなく，いくつか並べて書くこともできます．その時は，,（コンマ）で区切りますが，;（セミコロン）や空白で区切ることもできます．この区切りに用いる文字を，区切り記号と呼んでいます．

```
100 print "abc","def"
110 print "abc";"def"
120 print "abc" "def"
130 end
run
abc           def
abcdef
abcdef
Ok
■
```

セミコロンや空白で区切ると，2つの値は間隔を開けないで表示されます．コンマの場合は間隔を開けて表示するのですが，この表示の仕方にはちょっとした規則があります．それは，それぞれの値の先頭を14桁おきに表示することです．

```
100 print "1234567890123456789"
110 print "*","*"
120 print "///";"/"
130 print "**";"abc","*"
140 end
run
1234567890123456789
*             *
////
**abc         *
Ok
■
```

PRINT の画面への表示では，値や文字列を表示する領域が，14文字ごとに分けてとられており，区切り記号にコンマを使うと，次の表示領域の始めから値を表示するのです．区切り記号にセミコロンや空白を使うと，その領域は無視され，直前に表示したものの直後に表示が行われます．

ところで数値の表示の仕方は，少し変っていることに気が付きましたか．

```
100 print "1234567890123456789"
110 print "/";2;"/","*";-2;"*"
120 print -0.0000001234
130 print 1.234E21
140 end
run
1234567890123456789
/ 2 /         *-2 *
-1.234E-07
 1.234E+21
Ok
■
```

数値の前には，符号を表示するための空白が，1つとられてあり，数値が負の数の場合は－(マイナス)が表示されます．また数値の後には，必ず空白が1つおかれています．実数型の数値は，必ずその型の有効数字の桁数内で表示が行えるように，実数形式と指数形式のどちらかを選んで表示を行います．

PRINT は，式の値を全部表示すると改行を行いますが，この動作をさせないようにすることもできます．それには式の並びの最後に，コンマかセミコロンを書いておけばよいのです．こうすると改行は行われず，次の PRINT での表示が，同じ行で続けられます．

```
100 print "1234567890123456789"
110 print "abc";
120 print "def",
130 print "***"
140 end
run
1234567890123456789
abcdef        ***
Ok
■
```

区切り記号（空白を除く）を式の最後に付ければ，改行は行われません．それとは反対に，改行だけを行わせることもできます．それは，式を書かずに PRINT だけを使えばよいのです．

```
100 print "abc"
110 print "def"
120 print
130 print "***"
140 end
run
abc
def

***
Ok
■
```

セミコロンを使えば文字列などを続けて表示できますが，ちょっと注意しておきたいことがあ

ります．それは，表示する数値や文字列の長さが，その行のカーソルより後の余白よりも長い場合は，改行して次の行の先頭から表示が行われることです．次の例は1行80字で行ったものです．

```
new
Ok
100 a$="1234567890"
110 print a$;a$;a$
120 print 111111111#^2
130 end
run
123456789012345678901234567890
 1.2345678987654340D+16
Ok
■
```

なお文字列や変数をセミコロンや空白で区切った場合は，それらを省略することができます．

さて皆さん，PRINT の使い方が分かりましたか．PRINT は，数値や文字列などの式の値を画面に表示（出力）する働きをします．プログラムの中で何かメッセージを出したり，結果を表示させたりしたい時は，この PRINT を使えばよいことを覚えておいて下さい．

説明が最後になってしまいましたが，PRINT は？(疑問符，クエスションマーク）で代用できるようになっています．

```
new
Ok
100 ? "print = ?"
110 end
run
print = ?
Ok
list
100 PRINT "print = ?"
110 END
Ok
■
```

リストをとると，？は PRINT に変換されて表示されます．覚えておくと便利です．

この様な簡略形が使える命令が，他にも2つあります．1つは REM (後述) で，もう1つは代入文です．この代入文には，実は代入を行う命令が用意されているのです．それは LET という命令で，正式な代入文の書き方は次の様になります．

LET　A＝１２３

けれども，この LET を省略した形式が許されているのです．代入文はプログラムの中では頻繁に使いますから，このように省略してもよいようになっているのです．

☆INPUT（データの入力）

INPUT はキーボードから入力されたものを変数に代入する働きをします．使い方は，INPUT の後にキーボードからの値を代入する変数を，コンマで区切って並べればよいのです．

```
100 input a,b
110 print "a*b=";a*b,"a/b=";a/b
120 end
run
? ■
```

INPUT 文が実行されると，？（疑問符）が表示されキーボードからの入力待ちの状態となります．そこでキーボードから変数に代入する値を入力します．値を代入する変数が複数個ある時は，それぞれの値をコンマで区切って入力します．入力の終りはもちろんリターンキーです．

```
run
? 2,3
a*b= 6          a/b= .666667
Ok
■
```

もし，変数の数と入力された値の数が合わなかった場合は，警告メッセージが表示されて再び入力待ちの状態となります．

```
run
? 4.7
?Redo from start
? ■
```

上の例は 4 と 7 の値を入力しようとしたのですが，コンマをピリオドと間違えてしまったため，結果的には 1 つの値しか入力されなかったことになってしまったのです．こんな場合は，? Redo from start（最初からやり直せ）とメッセージが出て来る訳です．これは入力した値の数が，足りなかった場合ですが，多く入力し過ぎても，この警告が出力されます．

また，入力した値の型が変数の型と一致しない場合も，このメッセージは出力されます．例えばこの例では変数が数値型ですから，数値以外のものを入力するとこうなります．

```
run
? a
?Redo from start
? ■
```

それでは，この様な間違いを無くすにはどうすればよいでしょうか．それには何を入力すればよいのか分かるようにすればいいのです．前のプログラムの INPUT 文を次の様に変更して下さい．

```
100 INPUT "a,b";A,B
```

こうすると INPUT に続く文字列（これをプロンプト文といいます）が疑問符の前に表示されます．これならば，何を入力すればよいのかがはっきりするでしょう．

```
list
100 INPUT "a,b";A,B
110 PRINT "a*b=";A*B,"a/b=";A/B
120 END
Ok
run
a,b? 4,7
a*b= 28          a/b= .571429
Ok
■
```

このプロンプト文を表示させる時に,疑問符を表示させなくすることができます.プロンプト文に続くセミコロンをコンマに変えればよいのです.

```
100 input "a,b",a,b
run
a,b■
```

プロンプト文を表示させた場合でも,INPUT 文の動作は前に説明したことと全く同じです.

```
run
a,b-2,a
?Redo from start
a,b-2,3
a*b=-6          a/b=-.666667
Ok
■
```

次に,入力した値が,どのように変数に代入されるかを見てみましょう.数値変数の場合は,入力された数値は,その変数の型に変換されて代入が行われます.もしその値が,変数の型の範囲を越えた場合は,再度入力を促します.

```
100 input "a%";a%
110 print "a%=";a%
120 input "a!";a!
130 print "a!=";a!
140 input "a#";a#
150 print "a#=";a#
160 end
run
a%? abc
?Redo from start
a%? 3.14
a%= 3
a!? 123456789
a!= 1.23457E+08
a#? 1.2345678E21
a#= 1.2345678D+21
Ok
■
```

倍精度型変数での入力にちょっと注目して下さい.これを見ると,単精度型の数値を入力しても倍精度型に誤差なく変換されています.代入文で単精度型を倍精度型に代入すると誤差が生じたことを思い出して下さい.この点が,INPUT 文と代入文の違うところです.

さて,文字変数への文字列の入力の場合はどうでしょうか.INPUT の働きは,数値変数への入力の場合とほとんど同じです.文字変数が複数ある場合は,入力する文字列をコンマで区切り

ます．文字列は "(引用符) で囲む必要はありません．もし文字列の前後に空白があってもそれらは無視されます．これらの空白も入力したい場合は、引用符で囲む必要があります．また文字列の中にコンマを含んでいるものも引用符で囲まなければなりません．

```
new
Ok
100 input a$,b$,c$
110 print "/";a$;"/";b$;"/";c$;"/"
120 end
run
?    abc    ,"    abc    ","abc,def"
/abc/    abc    /abc,def/
Ok
■
```

引用符で囲めば，それ以外の文字は何でも入力することができます．

ところで，もし INPUT 文での入力の際に，何も入力せず，リターンキーを押した場合はどうなるでしょうか．これは 0 やヌルストリングが入力されたものとみなされます．

```
100 a=123
110 input a
120 print a
130 a$="abc"
140 input a$
150 print "/";a$;"/"
160 end
run
?
 0
?
//
Ok
■
```

この場合は，ただスペースだけを入力した場合と同じことです．ところで変数が複数ある場合は，必要な数のコンマは入力しなければなりません．

```
new
Ok
100 input "a,b,c";a,b,c
110 print a,b,c
120 end
run
a,b,c?
?Redo from start
a,b,c? ,,
 0             0             0
Ok
■
```

INPUT 文は，プログラムの実行を一時停止して，キーボードから入力された値を，変数に取り込む働きをします．代入文とは違って，変数に代入させる値を外部（キーボード）から入力することができますから，プログラム中で処理を行う値などを，自由に変えたりすることができます．INPUT は，プログラムの中で外部からデータを入力したい場合に使います．

☆GOTO（プログラムの流れの変更）

GOTO はプログラムの流れを指定した行に移す働きをします．今までのプログラムは，行番号の小さいものから始まって，最後の END 行で実行が終るものだけでした．これだとプログラムの流れは上から下へ一直線で，同じ仕事を何度か行うには，同じプログラムを何行も作るか，実行が終るごとに，再び RUN で実行させなければなりませんでした．もし，プログラムの流れがいつも次の行に移っていくのではなく，どこかほかの行に移すことができれば，１つの仕事をするプログラムを何度も実行させたりすることができます．GOTO を使えばプログラムの流れを他の行に移すことができます．

GOTO を使った例を見てみましょう．

```
100 input "a,b";a,b
110 print "a*b=";a*b,"a/b=";a/b
120 goto 100
run
a,b? 10,3
a*b= 30         a/b= 3.33333
a,b? ■
```

120行にあるのが GOTO 文です．「GOTO　行番号」と書くことによって，指定した行番号の行にプログラムの実行を移すことができます．この例ではプログラムの先頭に戻って，同じ仕事を何度も行うようになっています．

このように，プログラムの流れが同じ所を何度も回っているものを，ループと呼んでいます．このプログラムは無限にループを回ることになりますから，無限ループという訳です．無限ループは実行が永久に終わりません．この実行は外部から，強制的に中断させない限り止まらない訳です．外部からプログラムの実行を中断させるには，STOP キーを使います．このキーを押してみて下さい．

```
run
a,b? 10,3
a*b= 30         a/b= 3.33333
a,b?
Break in 100
Ok
■
```

以上の表示が行われて，コマンドレベルに戻りました，100という数字は，STOP キーが押された時に実行中であったプログラム行の行番号です．STOP キーを押せば，大抵の場合はプログラムの実行を中断できます．

無限ループの例として，分かりやすいのが次のプログラムでしょう．数字が１ずつ大きくなっていきます．オーバフローが生じるか，STOP キーを押さない限り止まりません．

```
100 i=0
110 i=i+1 : print i;
120 goto 110
```

```
run
 1  2  3  4  5  6  7  8 ^C
Break in 110
Ok
■
```

ところで110行を見ると，代入文と PRINT 文が：（コロン）で区切られて１つの行に続けて書いてあります．これはマルチステートメント（複文）といって，１つの行に複数の命令を書く方法です．いくつかの命令文をコロンで区切って書けばよいのです．それらの命令は前から順に実行されていきます．

次のプログラムを見て下さい．皆さんはこのプログラムをどう思いますか．

```
100  INPUT  A
110  GOTO  140
120  PRINT  B
130  END
140  B=A^2
150  GOTO  120
```

これはあまりよくないプログラムの例なのです．GOTO 文を使うと，プログラムの流れが飛躍しますから，あまり頻繁に使うとプログラムが大変見にくくなってしまいます．このような例も，プログラムが大きくなり，修正などを繰り返していると，しばしばできる形なのです．プログラムは，その流れが上から下へすっきりと流れているのが分かり易くて良いとされています．不用意に GOTO 文を使ってプログラムの流れを複雑にすることの無いように心掛けて下さい．

☆END（プログラムの終了）

END はプログラムの実行を終わらせる働きをします．この命令を実行するとプログラムの実行は終了しコマンドレベルに戻ります．

END 文は，プログラムの実行を終了させたい場所に置きます．また１つのプログラムに幾つ使っても構いません．なおプログラムリストの最後で実行が終わる場合は，そのEND文は省略することができます．

しかし，プログラムの最後の END 文はなるべく省略しない方が良いと思います．古いプログラムを消し忘れていたりして何かのプログラムがその行より後に残っていた場合などに，不用意にそれらが実行されるのを防ぐことができるのです．それともう１つ，END 命令が書いてあるということは，他人にも，ここがプログラムの終りであることが明確に伝わります．

END 文は，プログラムの実行が終了する所には，必ず置きましょう．

☆IF　THEN　ELSE（条件による分岐）

IF 文は条件判断を行って，プログラムの流れを変える働きをします．何らかの条件を表す式を

調べて，それが成立していればある仕事をし，成立していなければ前とは別の仕事をするのです．

この命令の使い方には，いくつかの形があります．一番基本的なものは次の形です．

IF　論理式　THEN { 文 / 行番号 }

論理式は条件判断を行うためのもので，その値は真（－1）か偽（0）のどちらかになります．この IF 文は論理式が真，つまり条件が成立しているならば，THEN の後の文が実行されます．もし論理式が偽，条件が成立していないならば，THEN に続く文は実行されません．THEN の後が行番号の場合は，条件が成立したならばその行番号の行にプログラムの実行が移ります．

ではこの IF 文を使って，前に出て来た無限ループのプログラムを，条件が成立したら終了するように，変えてみましょう．

```
100 i=0
110 i=i+1 : print i;
120 if i<8 then goto 110
130 end
run
 1  2  3  4  5  6  7  8
Ok
■
```

120行の IF 文の中にある論理式（I<8 のことです）を見て下さい．この式は，I が 8 未満の場合に真となります．そしてその時は THEN に続く GOTO 文を実行して110行にプログラムの実行が移ります．もし I が 8 以上になると論理式は偽となり，GOTO 文は実行されずプログラムの流れは次の行に移り，そして終了するのです．このプログラムは I が 8 より小さい間はループを形成し，I が 8 以上になるとループを抜け出すようになっています．それでは次のプログラムはどうでしょうか．

```
100 i=0
110 i=i+1 : print i;
120 if i>=8 then end
130 goto 110
run
 1  2  3  4  5  6  7  8
Ok
■
```

この結果は，前のプログラムと同じです．が，IF文の論理式の条件が前とは反対で，I が 8 以上になったならば THEN 以下の文を実行（プログラムが停止）するようになっています．

このように条件判断の考え方によって同じ内容のプログラムでも，書き方が随分違ってくることに気を付けて下さい．

皆さんはどちらのプログラムの方が分かりやすいですか．私は前のプログラムの方が，プログ

ラムの流れから見て分かりやすいのではないかと思います．

ところで前のプログラムの THEN の GOTO 文はただの行番号だけの形に，書き直せることはお分かりですね．しかし THEN の後に行番号があるのはどこか不自然です．こんな場合は，THEN の代りに GOTO を使うときれいになります．というのは THEN の後に行番号が続く場合は，THEN を GOTO に置き換えることができるからです．

```
120 IF I<8 GOTO 110
```

THEN の後には文か行番号が続くのですが，文の場合はマルチステートメントを使うことができます．これは条件が成立した時に，1つの命令だけでなく幾つかの命令を実行させることができる訳です．この方法を使って前のプログラムを書き直してみましょう．

```
new
Ok
100 i=0
110 if i<8 then i=I+1 : print i; : goto 110
120 end
run
 1  2  3  4  5  6  7  8
Ok
■
```

このようにも書ける訳ですが，ちょっとプログラムが見にくくなってしまいました．前のほうが良かったかもしれません．

さて今までの IF 文は，条件が成立した時に何か仕事を行うものでした．これは IF　×××　THEN △△△の書き方をすれば，「もし×××ならば△△△」と表現することができるでしょう．それでは，「もし×××ならば△△△さもなければ□□□」といった内容のものはどうすればよいでしょうか．つまり条件の成立，不成立に応じて別々の事を行わせたい場合です．例として何か数値を入力して，それが偶数ならば EVEN，奇数ならば ODD と表示するプログラムを考えてみます．今までのことから次のようなプログラムを作ることができるでしょう．

```
100 input a%
110 if a% mod 2 =0 then a$="even" : goto 130
120 a$="odd"
130 print a%;a$
140 end
run
? 2
 2 even
Ok
run
? 3
 3 odd
Ok
■
```

IF 文の論理式は，A％が偶数の時に真となります．この式の意味はお分かりですね．さて IF THEN 文は条件が成立してもしなくても結局は次の行に実行が移ってしまうので，条件が成立しなかった場合の処理（A％が奇数だった場合に行う仕事）を次の行に書いて，条件が成立したら

その行を飛ばして実行するような細工が必要です．そのため IF　THEN 文の最後に GOTO 文が必要なのです．このように IF　THEN 文で，条件によって２つの事を分けて実行する場合を表そうとすると，プログラムの流れが見にくくなってしまいます．この不都合を解消するには，「もし×××ならば△△△さもなければ□□□」のような形の IF 文が使えればよい訳です．それは次の形のものです．

IF　論理式　THEN $\left\{\begin{matrix}\text{文}\\\text{行番号}\end{matrix}\right\}$ ELSE $\left\{\begin{matrix}\text{文}\\\text{行番号}\end{matrix}\right\}$

この IF 文は，論理式が真ならば THEN に続く文だけを実行し，偽ならば ELSE に続く文だけを実行します．どちらも，その文の実行が終われば次の行にプログラムの流れが移ります．もし THEN や ELSE に行番号が続いているならば，その行に分岐します．この形を使えば前のプログラムがすっきりと表せるでしょう．

```
100 INPUT A%
110 IF A% MOD 2=0 THEN A$="even" ELSE A$="odd"
120 PRINT A%;A$
130 END
```

IF 文の THEN や ELSE の後には，どんな文を書いてもよく（もちろん，マルチステートメントも使えます），もう１つ IF 文を書くこともできます．こうすると，より複雑な条件判断を行わせることができます．例えば A と B の２つの変数があって，この２つの値の大小関係を求めたい場合があったとします．１つの例として次のプログラムが考えられます．

```
100 input "a,b";a,b
110 if a=b then a$="a=b" else if a>b then a$="a>b" else a$="a<b"
120 print a$
130 end
run
a,b? 2,3
a<b
Ok
■
```

こうすれば，２つの値の全ての場合分けが，１つの行で書けます．

ところで，１つの IF 文の中に幾つかの IF 文を使う場合に，注意しておかなければならない事があります．それは THEN と ELSE の対応です．IF 文は必ずしも IF　THEN　ELSE の形で用いられる訳ではなく，ELSE を使わない場合もありました．もし１つの IF 文で ELSE を省略した場合に，後から別の IF 文で ELSE が使ってあると，その ELSE が前の IF 文の THEN に対応するものとみなされてしまうのです．IF 文を多重（１つの IF 文の中にまた IF 文を用いること）にして使う場合は，よく注意して下さい．

IF 文を書く時に，THEN と ELSE の対応が分かりにくい時は，適当に空白を入れて書くとよいでしょう．プログラムの１行の長さが行番号を含めて255文字以内ならば，いくらでも空白を入

れて見やすくするのは構いません．皆さんの分かり易いように書くとよいでしょう．次のプログラムは前のものを空白を入れて見易くしたものです．

```
100 INPUT "a,b";A,B
110 IF A=B THEN A$="a=b"
          ELSE IF A>B THEN A$="a>b"
                      ELSE A$="a<b"
120 PRINT A$
130 END
```

IF 文は条件判断を行うためのもので，プログラムの流れをコントロールする働きをします．GOTO 文を使ったループを終わらせる条件を判断したり，ある値の場合分けをして，いろいろな処理を行ったり，様々に使われます．この IF 文の機能を十分に活用するためにも，条件判断を行う場合の条件を表す論理式の書き方には，十分に慣れておかなければなりません．論理式の使い方によっては，IF 文の使い方が随分違って来ます．例えば，変数Aの値が，3から7までの範囲（3と7を含む）にあるかを調べる IF 文を考えてみて下さい．皆さんはどうプログラムしましたか．

```
if 3<=a then if a<=7 then print "3<=a<=7"
```

このように書いた人は，あまり上手とは言えません．次のように書くのがうまいやり方です．

```
if 3<=a and a<=7 then print "3<=a<=7"
```

論理式の意味は分かりますね(分らない人は論理演算子の項を読み直しましょう)．論理式の書き方によっては IF 文が，随分単純になることが分かります．このように，条件判断を行う場合には，その条件を十分理解して適切な論理式を書かなければなりません．論理式の書き方によって，IF 文が随分変わってくるのです．皆さんも IF 文を使う際には条件の表し方に十分注意するようにして下さい．

3.1.2 プログラミングの流れ

プログラムを作るために，最低必要な命令についてお話ししましたから，今度はプログラムを作るための方法について述べていきましょう．

プログラムは命令の集まりであり，コンピュータはこれを順に実行して行きます．コンピュータに行わせたい仕事の手順を，命令を使って記述したものが，プログラムということです．この，コンピュータに行わせる仕事（処理）の手順をアルゴリズムと呼んでいます．コンピュータに何か仕事をやらせるには，そのアルゴリズムをまず考え，そしてそれを BASIC の命令を使って記述してプログラムにしていかなければなりません．

アルゴリズムを考えるには，まず仕事の内容をよく理解します．そしてその大まかな流れを摑み，段々と詳しくしていくのです．この仕事の流れを表すのによく利用されるものに，流れ図(フローチャート）があります．例えば，前に出て来た2つの変数の内容を比較するプログラムのフ

ローチャートは，次の様なものです．

フローチャート1

START

値の入力

比較

結果の表示

END

フローチャート2

START

A,Bに値を入力する

A=B,A>B,A<B の3つの関係を判断する

A,Bの関係を表示する

END

フローチャート3

START

INPUT A,B

A=B？ — YES → A$＝"A=B"

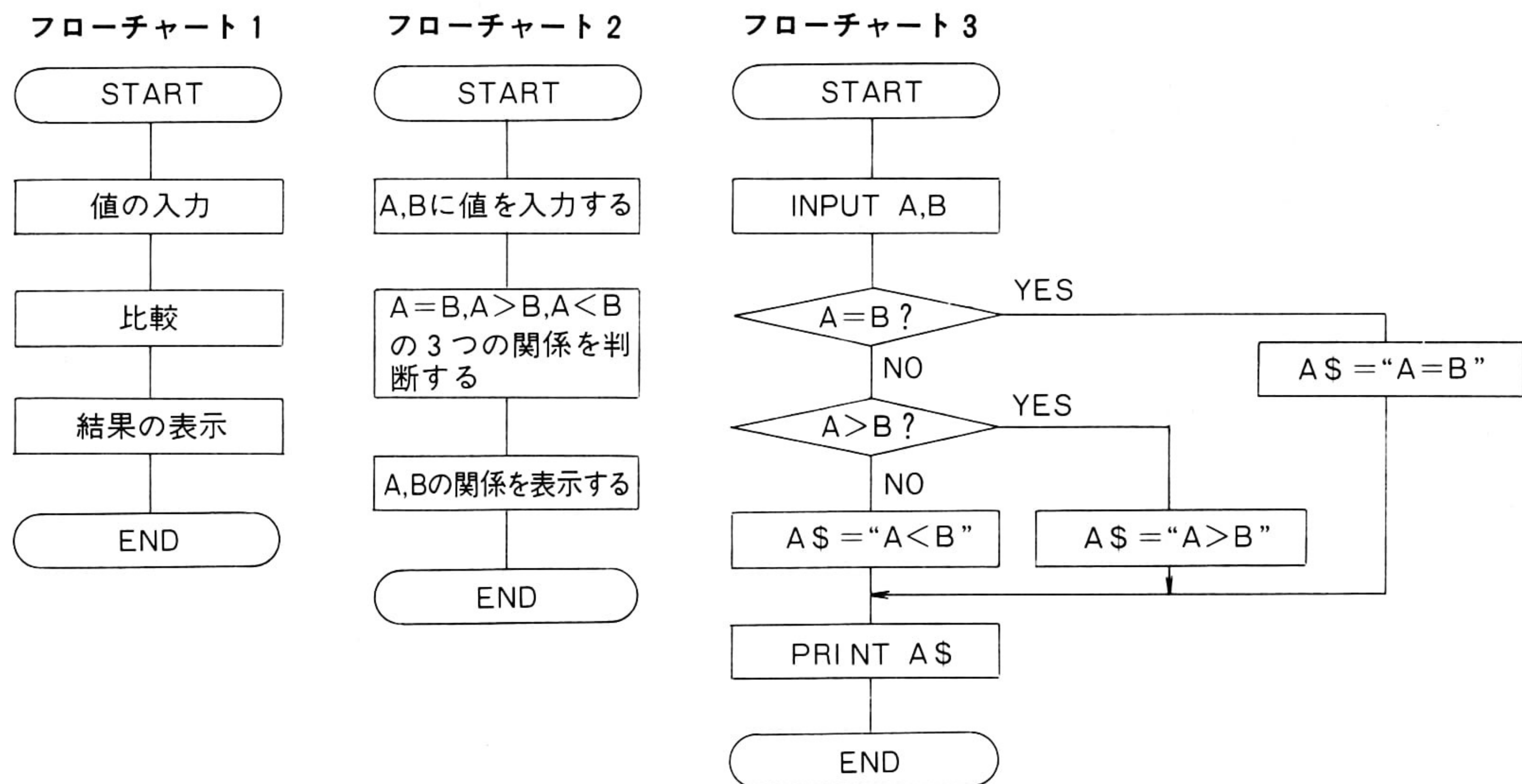
NO

A>B？ — YES → A$＝"A>B"

NO

A$＝"A<B"

PRINT A$

END

フローチャートはSTARTから始まってENDで終ります．四角の枠はいろいろな処理を表し，菱形は条件判断を行うことを表しています．そしてプログラムの流れはSTARTからENDの方向へ進み，条件判断の場合は指定された方向に分岐するのです．

さて例として示したフローチャート1は，大まかなプログラムの流れを表しています．最初はこの程度のものを考えて，段々と細かくしていきます．フローチャート3はかなり細かくアルゴリズムを表しています．これぐらい詳しいとすぐにプログラムを作ることもできるでしょう．

アルゴリズムを考える上でもう一つ重要なことは，プログラムの中で使用するデータをどのように表すかを決めることです．例えば変数は幾つ使うか，その型や変数名はどうするかなどを決めます．これはアルゴリズムを詳しくしていく過程で，使用するデータの種類をリストアップしていけば良いでしょう．

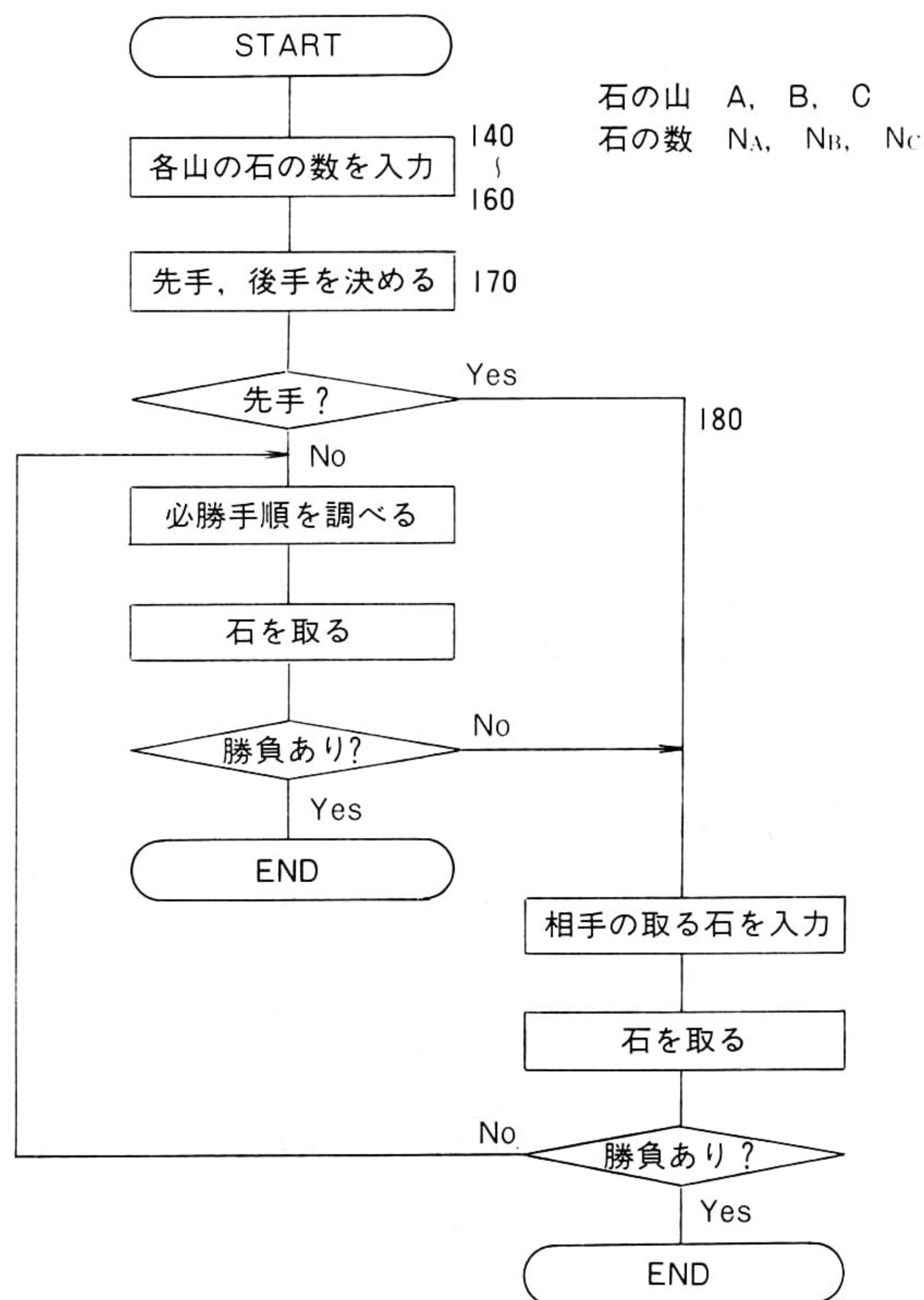
プログラムを作るためには，行わせたい仕事の内容をよくつかんで，その実現のための手順(アルゴリズム）を求めなければなりません．最初は大まかに，そして段々と細かくアルゴリズムを書いていきます．この時，フローチャートや皆さんの分かりやすい，他のどんな方法を使っても構いませんが，なるべく紙に書いて考えるようにしましょう．頭の中だけで考えると，とかくミスを犯しがちです(特に条件判断やプログラムが長くなる場合)．またアルゴリズムを求める時にプログラムで使用できる命令を考えて，プログラムに仕易いものを考えることが大切です．前に説明しました，データの入力，データの表示，条件判断，プログラムの分岐，終了などの機能をうまく使えるようにすることです．こうしてアルゴリズムを考えて，それをプログラムに直せば，プログラムは一応できあがりというわけです．後は，プログラムが正しく働くかをチェックして，誤りがあれば訂正を行います．正しく働くようになればプログラムは完成です．

3.1.3 サンプルプログラム

今までに説明してきた機能を使って，何か少し大きなプログラムを作ってみます．

ここでは NIM（三山くずし）というゲームプログラムを作ってみました．NIM（ニム）ゲームは2人で行うゲームです．ルールは，幾つかの石を積んだ山を3つ用意して，2人が交互に石を取っていき，最後に石を取った方が勝ちです．ただし石を取る数は自由ですが，一度に1つの山からしか取れません．

このゲームをコンピュータと対戦できるようにしてみました．人間とコンピュータが交互に石を取っていく勝負です．

プログラムのアルゴリズム（フローチャート）とプログラム本体を次に示します．

START
石の山　A，B，C
石の数　N_A，N_B，N_C
各山の石の数を入力　140〜160
先手，後手を決める　170
先手？　Yes　180
No
必勝手順を調べる
石を取る
勝負あり？　No
Yes
END
相手の取る石を入力
石を取る
勝負あり？　No
Yes
END

```
100 PRINT "********************"
110 PRINT "***** NIM GAME *****"
120 PRINT "********************"
130 PRINT
140 PRINT "ｿﾚｿﾞﾚﾉ ﾔﾏﾉ ｲｼﾉ ｶｽﾞｦ ﾆｭｳﾘｮｸｼﾃｸﾀﾞｻｲ｡"
150 INPUT "A,B,C:(2=< A,B,C <=15)";NA%,NB%,NC%
160 IF NA%<2 OR 15<NA% OR NB%<2 OR 15<NB% OR NC%<2 OR 15<NC%
     GOTO 140
170 INPUT "ｾﾝﾃ,ｺﾞﾃ:(Y:ｾﾝﾃ)";Y$
180 IF Y$="Y" OR Y$="y" GOTO 550
190 PRINT
200 NIM=NA% XOR NB% XOR NC%
```

```
210 IF NIM<>0 GOTO 250
220 IF NA%>=NB% AND NA%>=NC%
     THEN NM%=NA%:NA%=NA%-1:GOTO 340
230 IF NB%>=NA% AND NB%>=NC%
     THEN NM%=NB%:NB%=NB%-1:GOTO 410
240 IF NC%>=NA% AND NC%>=NB%
     THEN NM%=NC%:NC%=NC%-1:GOTO 470
250 IF NIM ¥ 8 THEN M=8:GOTO 290
260 IF NIM ¥ 4 THEN M=4:GOTO 290
270 IF NIM ¥ 2 THEN M=2:GOTO 290
280 M=1
290 IF ((NA% AND M)¥ M)=0 GOTO 360
300 NM%=NA%
310 NA%=NA%-1
320 NIM=NA% XOR NB% XOR NC%
330 IF NIM<>0 GOTO 310
340 PRINT "ｺﾝﾋﾟｭｰﾀ ﾊ A ｶﾗ";NM%-NA%;"ｺ ﾄﾘﾏｽ｡"
350 GOTO 480
360 IF ((NB% AND M)¥ M)=0 GOTO 430
370 NM%=NB%
380 NB%=NB%-1
390 NIM=NA% XOR NB% XOR NC%
400 IF NIM<>0 GOTO 380
410 PRINT "ｺﾝﾋﾟｭｰﾀ ﾊ B ｶﾗ";NM%-NB%;"ｺ ﾄﾘﾏｽ｡"
420 GOTO 480
430 NM%=NC%
440 NC%=NC%-1
450 NIM=NA% XOR NB% XOR NC%
460 IF NIM<>0 GOTO 440
470 PRINT "ｺﾝﾋﾟｭｰﾀ ﾊ C ｶﾗ";NM%-NC%;"ｺ ﾄﾘﾏｽ｡"
480 PRINT "A:";NA%,"B:";NB%,"C:";NC%
490 S%=NA%+NB%+NC%
500 IF S%=0 GOTO 690
510 IF S%<>1 GOTO 550
520 PRINT
530 PRINT "ｺﾝﾋﾟｭｰﾀﾉ ﾏｹﾃﾞｽ｡ ｻﾞﾝﾈﾝ･･･"
540 END
550 PRINT
560 PRINT "ｱﾅﾀﾉ ﾊﾞﾝ ﾃﾞｽ｡"
570 INPUT "ﾄﾙ ｲｼﾉ ﾔﾏ:(A,B,C)";M$
580 INPUT "ﾄﾙ ｲｼﾉ ｶｽﾞ";N%
590 IF N%<=0 GOTO 580
600 IF NOT (M$="a" OR M$="A") GOTO 620
610 IF NA%=0 GOTO 570
              ELSE IF NA%<N% GOTO 580
                              ELSE NA%=NA%-N%:GOTO 670
620 IF NOT (M$="b" OR M$="B") GOTO 640
630 IF NB%=0 GOTO 570
              ELSE IF NB%<N% GOTO 580
                              ELSE NB%=NB%-N%:GOTO 670
640 IF NOT (M$="c" OR M$="C") GOTO 660
650 IF NC%=0 GOTO 570
              ELSE IF NC%<N% GOTO 580
                              ELSE NC%=NC%-N%:GOTO 670
660 GOTO 570
670 PRINT "A:";NA%,"B:";NB%,"C:";NC%
680 IF NA%+NB%+NC%<>1 GOTO 190
690 PRINT
700 PRINT "ｱﾅﾀﾉ ﾏｹ｡ ﾊﾊﾊ･･･"
710 END
```

3.2 プログラミングに便利な機能及びステートメント

今までの解説は，プログラムを作るために最低必要な機能やステートメント（命令）についてでしたが，今度は知っていれば非常に便利なものについてお話しします．これらには，今まで幾つかの命令を組み合わせて実現していた仕事を1つの命令で行わせたり，今までに無かった機能を実現できる命令などがあります．上手に使えばプログラムの流れが分かり易く，そしてより複雑な仕事を行うプログラムが前よりも簡単に書けるようになります．そのためにも新しい機能や，どのように使えば便利なのかをよく理解して下さい．

3.2.1 ループ①（FOR～NEXT）

IF 文とGOTO 文を組み合わせてループを作ることができましたが，このループを回る回数があらかじめ分かっているような場合には，FOR～NEXT を代わりに使うことができます．

例えば，1からNまでの数を加えてその結果を求めるプログラムを考えてみましょう．IF 文を使ったループを考えると，次の様なプログラムとなります．

```
100 input n
110 i=0 : j=0
120 i=i+1
130 j=j+i
140 if i<n then goto 120
150 print j
160 end
run
? 10
 55
Ok
■
```

変数Iの値を1からNまで増加させて，値がNになった時にループを終了させています．このループを回る回数は，N回になることが初めから分かっているのです．こんな場合は，FOR～NEXTを使ってプログラムを書き変えることができます．

```
100 input n
110 j=0
120 for i=1 to n
130  j=j+i
140 next i
150 print j
160 end
run
? 11
 66
Ok
■
```

プログラムのFOR と NEXT で囲まれた部分がループの部分です．FOR 文は変数Iを1からNまで1ずつ増やしながら（このループを）繰り返し，Iの値がNを越えたならば，ループを抜け出して NEXT 以後に実行が移ります．変数Iはループを回った回数を数えて，ループの終了

条件をみるためのもので，このような変数をループのカウンタ（制御変数）と呼びます．

もう1つのFOR～NEXTを使ったプログラムを見て下さい．これは，前のプログラムと同じ結果が得られるものです．

```
100 input n
110 j=0
120 for i=n to 1 step -1
130  j=j+i
140 next i
150 print j
160 end
run
? 10
 55
OK
■
```

FOR文の終りのSTEPは，カウンタの値をどれだけ増減するかを指示しています．ここでは，−1が示されていますから，変数IはNから1ずつ減って，1までの値をとることになります．

以上のように，FOR～NEXTはプログラムのループを作るための命令です．FOR文はカウンタとして使用する変数とその初期値と終値，さらにその増分を定めます．そしてNEXTまでの文が実行され，カウンタに増分が加えられます．そしてカウンタの値が終値を越えていなければループの実行を繰り返し，そうでなければループを抜けてNEXT以後の文に実行が移ります．カウンタの増分が1の時は，STEPでの指定を省略することができます．

では次のプログラムを使って，FOR～NEXTの働きを実際に確かめてみましょう．

```
100 print "for i=L to M step N"
110 input "L,M,N";l,m,n
120 for i=l to m step n
130  print i;
140 next i
150 print " <";i;">"
160 goto 100
run
for i=L to M step N
L,M,N? 1,3,.5
 1  1.5  2  2.5  3  < 3.5 >
for i=L to M step N
L,M,N? 1,3,-1
 < 1 >
for i=L to M step N
L,M,N? ■
```

この例から分かるように，FOR～NEXTのループが終了した時は，ループで用いたカウンタの値は終値ではなく，必ず終値を越えていることに注意して下さい．また初期値と終値と増分の関係によっては，ループを1回も実行しない場合があります．それは次のような場合です．

①増分が正の値で，初期値が終値よりも大きい場合．

②増分が負の値で，初期値が終値よりも小さい場合．

どちらの場合もプログラムの実行はNEXTの次の文に移ります．この時のカウンタの値は初期値となっています．

他にも FOR～NEXT を使う上で注意して頂きたい事が幾つかあります．その1つは，FOR と NEXT は必ず一対一に対応して使わなければならないことです．例えば次のプログラムを見て下さい．

```
100 INPUT N
110 FOR I=1 TO 30
120  IF I MOD N=0 THEN NEXT I
130  PRINT I;
140 NEXT I
150 END
```

このプログラムは，指定した数の倍数を除いた1から30までの数の列を表示させるものです（あまり意味のあるプログラムとは言えませんが）．1つの FOR 文に対して，場合によって2つの NEXT 文のどちらかを対応させてループを作っています．しかしこのような書き方をするとプログラムは正常に働きません（エラーとなります）．1つの FOR 文には 必ず1つの NEXT 文しかプログラム上では対応して書くことができないのです．ですからこの場合は，IF 文で使った NEXT 文を GOTO 文に変えて，140行の NEXT 文に分岐するようにすればよいのです．またこれとは逆の，1つの NEXT 文に2つの FOR 文が対応するような使い方も許されていません．

また FOR～NEXT 文のカウンタの値は，プログラムの中で変えることができます．一方，初期値や終値，増分などは後から変えることはできません．このカウンタの値を変更することは，FOR～NEXTループの実行回数を強制的に変える場合に使われることがあります．カウンタの値を終値より大きくしてやれば，次の NEXT 文でループから抜け出させることができるのです．次のプログラムがその一例です．

```
100 input n
110 for i=1 to 30
120  if i=n then i=31
130  print i;
140 next i
150 end
run
? 6
 1  2  3  4  5  31
Ok
■
```

しかしこのようなことは，FOR～NEXT ループでやるべきではありません．FOR～NEXT 文はループの回数が予め分かっている場合に用いるものです．ループの回数が場合によって変るようなものには，そのための便利な命令が用意されています．

ところでループの中にまた別のループがあるような場合には，FOR～NEXT ループを多重にして使うことができます．このループなどが多重になることを，ネスティングと呼んでいます．

```
100 for i=1 to 5
110  for j=1 to i
120   t=i+j
130  next j
140  print t;
150 next i
160 end
```

```
run
 2   4   6   8   10
Ok
■
```

FOR～NEXT のネスティングは，１つの FOR～NEXT の中に完全に他の FOR～NEXT が入らなければなりません．つまり入れ子になっていなければならないのです．2つの FOR～NEXT ループが，オーバーラップするようなことは許されません．また，それぞれのループの制御変数（カウンタ）には別の変数を使わなければなりません．ところで前のプログラム例は大変無駄のあるプログラムなので，アルゴリズムを考えてすっきりと直してみて下さい．

○良い例

```
FOR  I＝1  TO  9
    FOR  J＝1  TO  9
        ・
        ・
        ・
    NEXT  J
NEXT  I
```

○悪い例

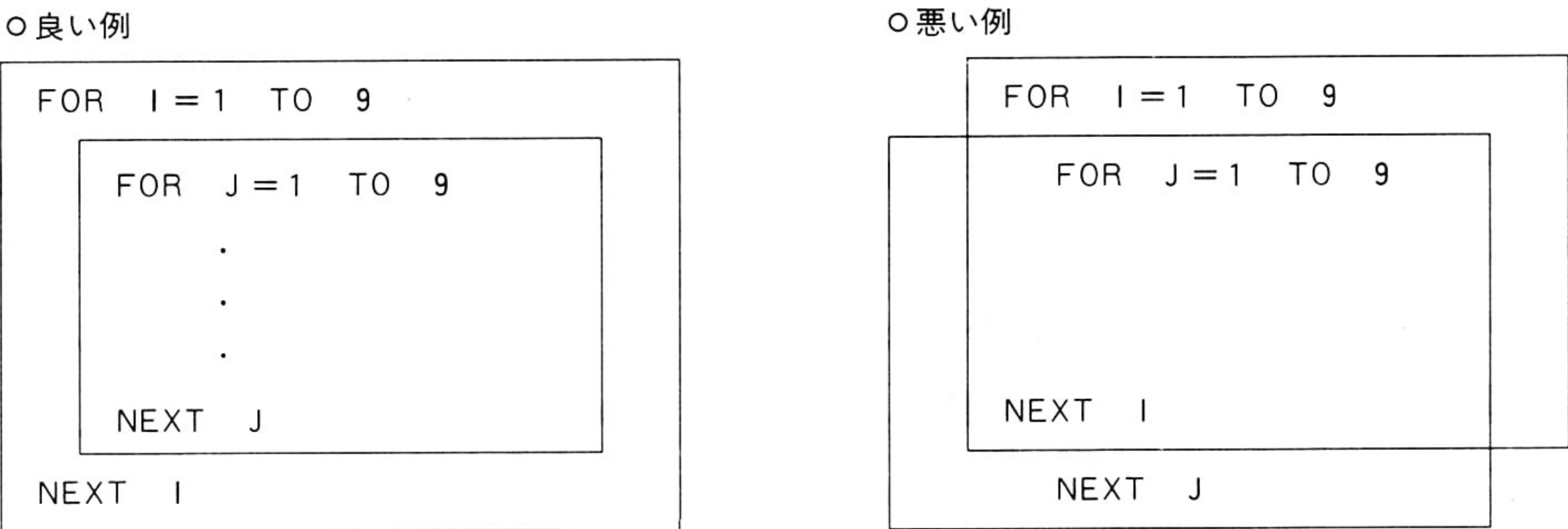
```
FOR  I＝1  TO  9
    FOR  J＝1  TO  9

NEXT  I
    NEXT  J
```

＊１つのＦＯＲ～ＮＥＸＴの箱（ループ）の中に完全に他のＦＯＲ～ＮＥＸＴの箱（ループ）が含まれている．

図１　入れ子構造

NEXT 文の変数名は，省略しても構いません．もちろん，ループの場合は必ず入れ子構造になっていなくてはなりません．

```
100 for i=1 to 4
110  for j=1 to 5
120   print i*j;
130  next
140  print
150 next
160 end
run
 1   2   3   4   5
 2   4   6   8   10
 3   6   9   12   15
 4   8   12   16   20
Ok
■
```

なお次の例のように，複数のループが同じ所で終わっているような場合は，NEXT 文を１つにまとめることができます．

```
new
Ok
100 for i=3 to 1 step -1
110  for j=1 to 2
120   print i*j;
130 next j,i
140 end
```

```
run
 3  6  2  4  1  2
Ok
■
```

この場合の変数名の並びは，内側のループのカウンタの変数名から順に並べます．NEXT 文を1つにまとめた場合は，変数の並びを省略することはできません．

このような FOR～NEXT ループは，ループの回数があらかじめ決まっている場合であれば，すっきりとしたプログラムが書けますし，なによりも速度の向上につながります．

3.2.2 ループ②(WHILE～WEND)

この命令もループを作るためのものです．ただし FOR～NEXT ループのようにループの回数が決まっている場合ではなく，ある条件が満たされている間ループを回るという働きをします．ただしこの命令は N_{88}-DISK BASIC でしか使えませんから注意して下さい．

例として，整数の除算を行うプログラムを考えてみます．2つの数A，Bを入力して，A¥Bの結果を他の方法で求めるものです．まず IF 文を使ったプログラムを示します．

```
100 input "a,b";a%,b%
110 i=0
120 if a%<b% then goto 160
130 a%=a%-b%
140 i=i+1
150 goto 120
160 print "a¥b=";i
170 end
run
a,b? 5,2
a¥b= 2
Ok
■
```

AからBの値を引いていって，Aの値がBよりも小さくなるまでにループを回った回数が答になります．ループを何回，回るかが予め決まっている(FOR～NEXT ループの場合は，初期と終値，増分によってループを回る回数がおのずと決まる）のではなく，ある条件が成り立っている間，何回かループを回るようになっているのです．それではこのプログラムを，WHILE～WEND 文を使って書き直してみます．

```
100 input "a,b";a%,b%
110 i=0
120 while a%>b%
130  a%=a%-b%
140  i=i+1
150 wend
160 print "a¥b=";i
170 end
run
a,b? 7,3
a¥b= 2
Ok
■
```

WHILE〜WEND 文は，WHILE 文の論理式が真である間は，WHILE 文と WEND 文の間にある文の実行を繰り返します．実際の動作は，まず WHILE 文で論理式の値を調べ，真ならば WHILE 文以後の WEND 文までを実行し，また WHILE 文まで戻って来ます．もし論理式の値が偽ならば，WEND 文以後の文にプログラムの実行が移るのです．もちろん，最初から論理式の値が偽であれば，ループは一度も実行されません．このように WHILE〜WEND 文は，ある条件が成り立っている間ループを回るプログラムを作るのに大変便利です．ただし条件の指定を間違えたりすると，GOTO 文を使った無限ループと同じになってしまいますから，注意が必要です．

WHILE〜WEND ループも多重にして使うことができます．この場合もそれぞれの WHILE〜WEND 文は，入れ子の形になっていなければなりません．

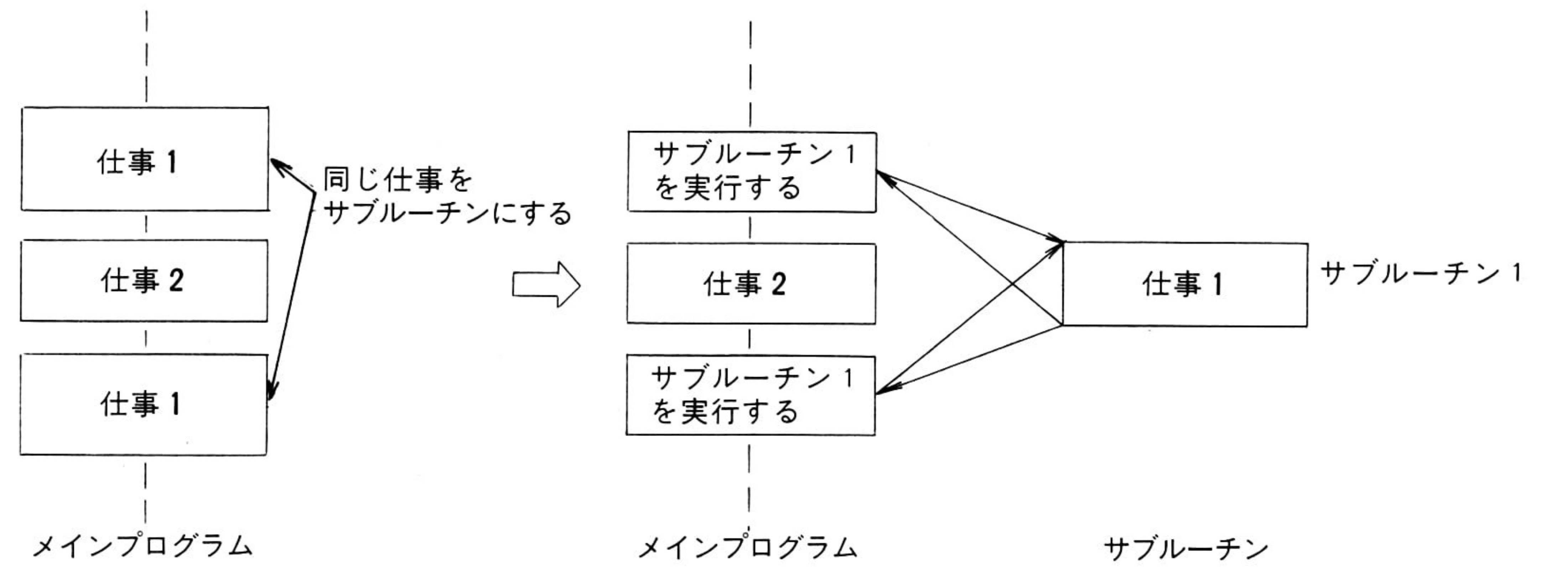
```
100 i=0
110 while i<4
120  i=i+1 : j=i
130  while j<10
140   print j;
150   j=i+j
160  wend : print
170 wend
180 end
run
 1  2  3  4  5  6  7  8  9
 2  4  6  8
 3  6  9
 4  8
Ok
■
```

WHILE〜WEND 文を FOR〜NEXT 文と組み合わせて多重ループを作ることも，もちろんできます．ただし，どんなときでも，多重ループを作る場合は，互いのループは必ず入れ子の形になっていなければなりません．

以上説明して来ました FOR〜NEXT 文と WHILE〜WEND 文を使えば，大抵のループは表すことができます．IF 文と GOTO 文を使ったループはあまり分かりやすいものではありませんし，間違いも多くなります．特に多重ループの場合はなおさらです．ループを作る場合はなるべく GOTO 文を使わず，FOR〜NEXT や WHILE〜WEND のループを使うようにしましょう．ただし，その場合もループの働きをよく摑んで，どちらか適切な命令を使うように注意して下さい．

3.2.3 サブルーチン(GOSUB〜RETURN)

プログラムの中で，同じ仕事を続けて繰り返して行わせるには，ループの形を使うことができました．それでは同じ仕事をプログラムのところどころで行わせたい場合には，どうすればよいでしょうか．プログラムに行わせる仕事が複雑になって来ると，プログラムの中で同じ仕事を行っている部分が，ところどころに現われてくることがよくあります．その場合に，同じ仕事を行うプログラムをその都度書いていたのでは，プログラム全体が大きくなってしまい，何か無駄な

感じもします．このような場合にはサブルーチンを使うことができます．プログラムの中で同じ仕事をしている部分を取り出してサブルーチンにしておけば，プログラムの中で必要な時にそのサブルーチンの仕事をさせることができるのです．

それでは実際にサブルーチンを使ったプログラムを見てみましょう．これはサブルーチンの使い方と，その実行のされ方を示したものです．

```
new
Ok
100 print "start"
110 gosub 1000
120 print "main program"
130 gosub 1000
140 print "program end"
150 end
1000 print "executed subroutine"
1010 return
run
start
executed subroutine
main program
executed subroutine
program end
Ok
■
```

1000行から始まるプログラムが，サブルーチンです．サブルーチンの終わりはRETURN文で示されています．プログラム本体（メインプログラム）からサブルーチンを実行させるには，GOSUB文を使います．GOSUB文は指定された行番号から始まるサブルーチンを呼び出す（サブルーチンコール）働きをします．GOSUB文で呼び出されたサブルーチンは，RETURN文が実行されると，呼び出したGOSUB文の次の文に実行が戻ります．GOSUB文を使えば，サブルーチンを自由に呼び出して実行させることができるのです．そして，サブルーチンの終わりはRETURN文でメインプログラムに戻ります．

サブルーチンは，プログラム中に幾つあっても構いません．これらのサブルーチンは行番号で

区別して呼び出せるのです．また，プログラム中でメインプログラムとサブルーチンの部分は，はっきりと区別しておかなければなりません．プログラムの例でもメインプログラムはENDで終わっており，サブルーチンにそのまま実行が移ることを防いでいるのです．サブルーチンを GOSUB 文で呼び出さずにそのまま実行するとエラーが生じてしまいます．

サブルーチンは，同じ仕事をプログラム中で何度か行っている場合に使うと，プログラムを短くすることができます．サブルーチンで行わせる仕事が長い場合はいっそう効果的です．またサブルーチンを使うことによって，別の利点も生じます．それはサブルーチンをある目的の仕事をするようにまとめれば，GOSUB 文はその仕事を一度に行うことができる新しい命令と考えることができるのです．複雑な仕事もサブルーチンにまとめて，メインプログラムから GOSUB で呼び出すようにすれば，プログラムの流れがすっきりと分かりやすいものになります．

サブルーチンは多重にすることができます．これは1つのサブルーチンの中から他のサブルーチンを呼び出すことです．プログラムの中で行われる仕事を細かく整理してサブルーチンにし，その小さなサブルーチンを集めてあるまとまった仕事をするサブルーチンを作るということができるのです．プログラムを作る際はプログラムで行わせる仕事をよく理解して，まとまった仕事はサブルーチンにしていき，そのサブルーチンを組み合わせて大きなプログラムにしていくやり方をすれば，プログラムも作り易く，その構造も分かり易いものとなります．

3.2.4 注釈文(REM)

プログラムの中にタイトルとか，ちょっとしたコメントなどを書くことができたら便利だと思いませんか．REM 文を使うとプログラムの中に自由にメッセージを書くことができます．使い方は，REM の後に続けて文章を書けばよいのです．また，REM の代わりに '(アポストロフィ)を使うこともできます．

```
100 REM TITLE
110 REM Sample Program
1000 REM Subroutine
```

REM 文を使ってプログラムで行っている仕事の内容や，サブルーチンの働きなどを書いておけば，プログラムを見た時にその構造や，何を行っているのかがよく分かります．

```
100 ' TITLE
110 ' Sample Program
1000 ' Subroutine
```

REM 文は実行されない文（非実行文）ですから，プログラムのどこにおいてもプログラムの実行に影響はありません．ただし，その行番号を GOTO 文などの飛び先に使うことはできます．

REM 文を使う時に注意しなければならないことが1つあります．それは，REM 文の後をコロンで区切ってマルチステートメントにはできないことです．

```
new
Ok
100 ' TITLE : print "title"
110 end
run
Ok
■
```

REMは，その後に続く文字の列を全てコメント文とみなしてしまうのです．反対に，マルチステートメントの行の最後に使うことはできます．この時は，区切りのコロンが省略できます．

プログラムの中ではREM文を使って必要に応じたコメント文を書いて，プログラムの流れや構造を分かり易くしておくとよいでしょう．後でプログラムを見直す場合に分かり易くなります．

3.2.5 ラベル

皆さんはGOTO文やGOSUB文を使う場合に，プログラムの分岐先の指定を行番号で行わなければならないのは，不便だと思いませんか．行番号はBASICがプログラムを実行するために使われるもので，私たちにとってはただの数字の並びであり，あまり意味のあるものではありません．GOSUB文でサブルーチンを呼び出す場合も，ただ行番号で指定されるだけで，何のサブルーチンなのかは，その行番号の所のサブルーチンを見てみなければ分からないのです．そこで何か意味のある名前でサブルーチンなどの行番号を指定できれば，大変便利だと思いませんか．そこでN_{88}-BASICでは，行番号の指定にラベルを使うことができるようになったのです．何はともあれ，ラベルを使ったプログラムを見て下さい．

```
100 *START
110 PRINT "label sample"
120 GOTO *START
```

100行の*STARTがラベルです．120行のGOTO文のように，行番号の代りにラベルを書けば，ラベルの付いた行の行番号を指定したのと同じになるのです．こうすればGOTO文の飛び先も，GOTO文を見ただけでどんな意味の所に飛んでいるかが分かるでしょう．

ラベルは，私たちがプログラム中でGOTO文などの分岐先の行を示すために，行に自由に付けることのできる名前という訳です．ラベルの付け方には，次のようなルールがあります．

ラベル名の先頭は必ずアスタリスク（*）で始まらなければなりません．またアスタリスク以後の名前は，変数名の付け方と同じです．長さに制限はありません（ただし1行に書ける範囲に限ります）．

ラベルは必ず行の先頭に付けなければなりません．またラベルの後にコロンまたはスペースで区切ってマルチステートメントとして書くことができます．

同じラベル名のラベルを重複して付けることはできません．

またこれらのラベル名は，LISTなどのコマンドでも行番号の代わりとして使うことができます．

皆さんもラベルを十分活用して，分かり易く見易いプログラムを作って下さい．

3.2.6 データの読み込み(READ, DATA, RESTORE)

プログラム中で変数にデータを入力するには，代入文かINPUT文を使うことができました．しかし代入文ではあまり多くのデータを扱うのは大変ですし，INPUT 文はその都度キーボードから入力しなければなりません．そこでデータを入力するための方法がもう1つあります．これはプログラム中にデータを用意しておいて，1つずつ変数に読み込んでいくのです．そのためには，READ 文と DATA 文を使えばよいのです．

READ文はDATA文に用意されたデータを1つずつ取り出して，指定された変数に代入していく働きをします．一方，DATA文は，READ文で読み込むための，数値や文字型などのデータを，格納しておく働きをします．次のプログラムを見て下さい．

```
100 for i=1 to 3
110  read a,b,c
120  print a,b,c
130 next i
140 end
150 data 1,2,3,4,5,6
160 data .7,-8,-90
run
 1             2             3
 4             5             6
 .7           -8           -90
Ok
■
```

110行がREAD文で，150行からのDATA文よりデータを読み込みます．FOR～NEXTループによってREAD文が実行される毎に，DATA文からデータを読み込んでいます．実行結果から分かるようにDATA文のデータは順に読み込まれており，同じデータが重複して読まれることはありません．またデータが2つ以上のDATA文に渡って置かれていても，それらは行番号の若いDATA文から順に読み込まれていきます．BASICはデータをどこまで読んだかを憶えていて，READ文が現れるごとに新しいデータを読み込んでいくのです．

READ文でデータを代入する変数はどんな型のものでも構いません．ただし数値変数に文字定数を読み込むようなことはできません．また数値データを読み込む場合は，データは変数の型に変換されて代入されます．

もしREAD文でデータを読み込んでいる内に，DATA文のデータの数が足りなくなった場合は，エラー（Out of data）となります．もちろん，READ文でデータを読み残してもエラーとはなりません．次にREAD文が現れた時に読み込まれるだけです．

DATA文には，コンマで区切って数値や文字列などのデータを書きます．1行に入る範囲ならば幾つでも書くことができます．文字定数は引用符（"）で囲まなくても構いませんが，文字列の前後に空白を含んでいたり，文字列の中にコンマがあるような場合は，引用符で囲まなければなりません．

```
100 read a%,a!,a#,a$
110 print "a%=";a%
120 print "a!=";a!
130 print "a#=";a#
140 print "a$=";a$
150 end
160 data 1.23456789,1.23456789
170 data 1.23456789,1.23456789
run
a%= 1
a!= 1.23457
a#= 1.23456789
a$=1.23456789
Ok
■
```

またDATA文は非実行文（PRINT，INPUT命令などのように即動作を起こさない文）ですから，プログラムのどこにでも置くことができます．その時も，データが読み込まれる順序は行番号の若い方からです．

ところで，一度読み込んだDATA文のデータを，再び初めから読み直すことができます．それには RESTORE 文を使います．

```
new
Ok
100 read a,b,c
110 print a,b,c
120 restore
130 read a,b,c
140 print a,b,c
150 end
160 data 1.2,-3.4,.567,8.9E21
run
 1.2           -3.4            .567
 1.2           -3.4            .567
Ok
■
```

RESTORE 文は，この次から READ 文が DATA 文の最初からデータを読み込むように働きがあります．また RESTORE 文で行番号を指定すると，指定された行番号以後の DATA 文からデータを読ませることができます．RESTORE 文を使えば，DATA 文のデータを何度でも READ 文で読むことができるのです．

RESTORE 文に行番号を付けた場合の例を示します．

```
100 read a,b,c
110 print a,b,c
120 restore 170
130 read a,b,c
140 print a,b,c
150 end
160 data 1.23,-3.45
170 data 0.0001234,1.23E2
180 data 1234567,32
run
 1.23          -3.45           1.234E-05
 1.234E-05      123            1.23457E+06
Ok
■
```

このように READ 文と DATA 文は，プログラム中で使用するデータを用意しておき，必要に応じて，それを読み込んで使うことができるのです．

3.2.7 配列

今までデータを取り扱うには変数を用いてきました．この変数は，変数名によってその値を参照することができるので，データを取り扱うには大変便利です．それでは大量のデータを取り扱う場合はどうでしょうか，この場合は今までの変数の使い方ですと，1つのデータごとに別の変数が必要となり，変数名も大変多くなってしまいます．これでは変数名を管理するだけでも大変で，プログラムも複雑になってしまいます．

そこで，大量のデータをまとめて取り扱おうというのが，配列です．配列とは変数の仲間なのですが，今までのものとは違って，1つの変数名で幾つかの変数の集合を表すことができるのです．そしてその集合の中の1つ1つの変数は別の方法で表します．例えば，配列は住所の指定に似たような形をしています．町名で配列の名前を表し，番地で配列の中の各変数を表すと考えることができるのです．

配列を表すには，普通の変数名にカッコでつくった式（数字）を付けた形を使います．

```
100 dim a(15)
110 for i=0 to 15
120  read a(i)
130 next i
140 for i=0 to 15 step 2
150  print "a(";i;")=";a(i),
160  print "a(";i+1;")=";a(i+1)
170 next i
180 end
190 data 1,1,3,3,5,5,7,7,9,9
200 data 10,10,12,12,14,14
run
a( 0 )= 1       a( 1 )= 1
a( 2 )= 3       a( 3 )= 3
a( 4 )= 5       a( 5 )= 5
a( 6 )= 7       a( 7 )= 7
a( 8 )= 9       a( 9 )= 9
a( 10 )= 10     a( 11 )= 10
a( 12 )= 12     a( 13 )= 12
a( 14 )= 14     a( 15 )= 14
Ok
■
```

100行の DIM 文は，配列をこれから使うよということを宣言するものです．120行や150行で使われているA(I)，160行のA(I＋1)というものが配列です．プログラムの中で，どれだけの変数が使われているか分かりますか．(A(I)と表した配列のIの値は，0から15まで変わります．実にA(0)からA(15)までの16個の配列変数が使われているのです．同じ名前（ここではA）でカッコの中の値を変えるだけで，たくさんの変数を取り扱うことができるのです．これが配列の特徴です．このA(0)，A(1)などを配列Aの要素といい，カッコの中の変数Iや，1，2，3などの数字を配列の添字と言います．添字は普通0から始まります．

DIM　A(15)　〔A(I)：(0≦I≦15)〕

配列A

A (0)
A (1)
A (2)
・ ・ ・
A (14)
A (15)

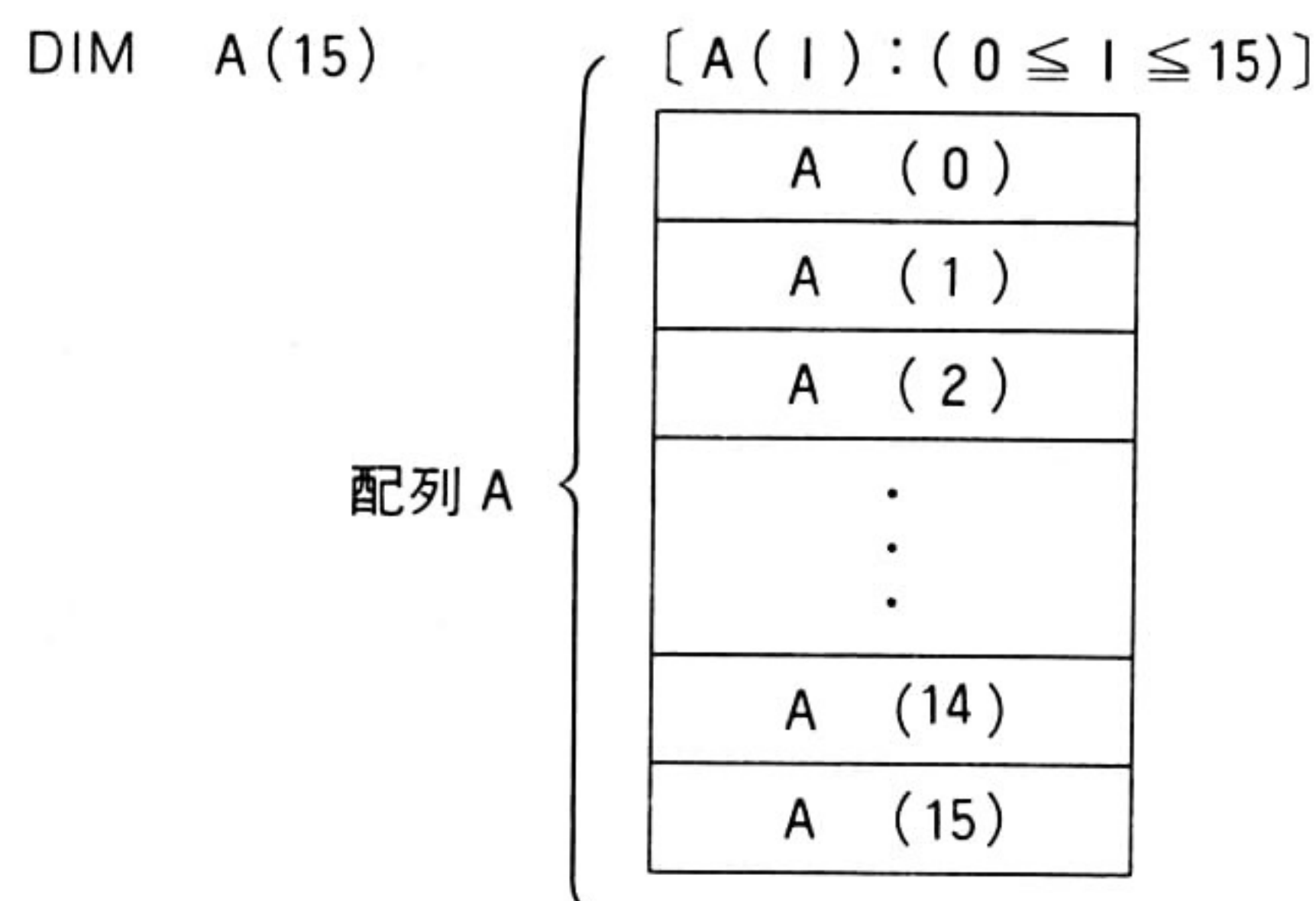



図2　配列変数

配列の各要素の働きは，普通の変数（単純変数）と全く変わりません．もちろん，このプログラムも単純変数も使って書くこともできます．ちょっとプログラムを書いてみましょう．

```
100 read a0,a1,a2,a3,a4
110 read a5,a6,a7,a8,a9
120 read a10,a11,a12,a13
130 read a14,a15
140 print "a0=";a0,"a1=";a1
150 print "a2=";a2,"a3=";a3
160 print "a4=";a4,"a5=";a5
      .
      .
      .
```

ああもう書くのがいやになりますね．御覧のように単純変数の場合は，変数１つ１つの別の名前で使ってやらなければならないため，随分面倒になってしまいます．配列の場合はA(Ｉ)の，Ｉを変えるだけでA(１)にもA(10)にもなることができるので，プログラムがとても簡単になっているのです．

さて前の配列を使ったプログラムの100行を見て下さい．このDIM文は，Aという配列を15までの添字で使うよ，ということを宣言しています．配列を使うには，まず配列の要素のための記憶領域を予め用意しておかなければならないのです．DIM文は，これから使う配列を用意させるための命令です．

DIM文で配列を宣言するには，DIMの後に使用したい配列の最大の添字を持ったものをコンマで区切って並べればよいのです．例えば，整数型の配列のＩ％をＩ％(０)からＩ％(７)まで，文字型の配列C$をC$(０)からC$(12)まで使いたい場合は，次のように書きます．

```
100 DIM I%(7),C$(12)
```

DIM文で配列の大きさを宣言すると，後で変更することはできません．同じ配列名で，２度配列を宣言することはできないのです．また，配列の添字がDIM文で宣言した範囲を越えるとエラーとなります（添字には負の値を使うこともできません）．

ところで，今まで使って来た配列は，みんな添字が１つのものでした．これを１次元配列と呼びます．しかし，配列では添字を幾つ持ってもよいのです．配列の添字が２個以上のものを，多次元配列と呼びます．

ここで２次元配列を考えてみましょう．添字の数が２つの配列です．これまでの１次元配列は，頂度要素が一列に並んでおかれていると考えることができます．それに対して，２次元配列は，要素が縦と横に行列をなしているものと考えることができるでしょう．ですから添字によって配列の要素を指定することは，要素の行列の何行目の何列目というような意味で行えばよいのです．例えば，Ａ(Ｉ，Ｊ)という配列は，要素の行列のＩ行目のＪ列目を表しているという訳です．

DIM A (M, N)　(0≦I≦M, 0≦J≦N)

A(0, 0)	A(0, 1)	・・・	A(0, J)	・・・	A(0, N)
A(1, 0)	A(1, 1)	・・・	A(1, J)	・・・	A(1, N)
A(2, 0)	A(2, 1)	・・・	A(2, J)	・・・	A(2, N)
⋮	⋮	⋮	⋮	⋮	⋮
A(I, 0)	A(I, 1)	・・・	A(I, J)	・・・	A(I, N)
⋮	⋮	⋮	⋮	⋮	⋮
A(M, 0)	A(M, 1)	・・・	A(M, J)	・・・	A(M, N)

図３　２次元配列

配列の添字の働きは，座標の働きとよく似ています．１次元配列は１次座標，２次元配列は２次元座標の考え方で配列の要素を指定できるでしょう．また，この考えを発展させれば，さらに高次元の配列も考え易くなります．

プログラム中で使われた配列が不要になった場合，その配列は抹消して，別の配列を宣言するようなことができます．不要になった配列を抹消して，新しく使う配列を宣言すれば，配列の記憶領域を節約できるのです．配列を抹消する働きをするのがERASE文です．

```
100 DIM A(8)
110 ERASE A
120 DIM A(12)
140 END
```

ERASEの後に抹消したい配列名を書きます．そして，ERASE文により配列を抹消した後，DIM文で再び配列を宣言します（配列を抹消せずに同じ名前の配列を宣言するとエラーとなります）．このことを利用すると，配列の要素を全てクリア（数値型配列は０に文字型配列はヌルストリングに）することが簡単に行えます．FOR～NEXTループを使っていちいち配列の要素をクリアしなくてもよい方法があるのです．例えばDIM A(50)で宣言した配列をクリアしない時は次のようにプログラムすればよいのです．

```
        .
        .
        .
210 ERASE A
220 DIM A(50)
        .
        .
        .
```

DIM 文で配列を宣言すると，その配列の要素は全てクリアされているのです．

注)BASIC では，添字の値が10以下の配列を使う場合は，DIM 文でその配列を宣言しなくても使うことができます．これはプログラム中で添字が10以下の配列が使われると，BASIC は自動的にその配列を宣言してくれるのです．例えばA(2)という配列が宣言されずに用いられたならば，BASIC は自動的に DIM A(10)を実行するのです．これは便利なことと思われるかもしれませんが，不用意にこれを行うと配列の記憶領域を無駄に使うことになります．つまりあなたがA(0)からA(5)までの配列しか使わないとしても，しっかりとA(10)までの配列が用意されてしまうのです．さらに例えれば，A(0，0)からA(1，1)までの配列を宣言せずに使ったとしたら，それ以外の11×11－4＝117個もの配列は無駄になっているのです．ですから配列は必要なだけの数を宣言して用いるようにして下さい．

また配列を自動宣言によって使うと，そのプログラムの中で幾つかの配列を使っているのかが，はっきりとは分からなくなってしまいます．これはプログラムの内容が分かりにくくなる原因になりますから，見易いプログラムを作るためにも配列は宣言して使うことを心掛けて下さい．

また，配列の添字は0から始まりますが，これを1から始まるように変更することができます．これにはプログラムの中で次の命令を実行させます．

```
OPTION BASE 1
```

これを実行すると，配列の添字は0から使うことができず，1から始まるようになります．ただしこの宣言は，DIM 文で配列を宣言したり，プログラム中で配列を用いた(配列の自動宣言が行われる）後に行うことはできません．

プログラムで使っている変数（単純変数や配列変数）をすべて抹消することができます．それには CLEAR 命令を使えばよいのです．CLEAR が実行されると，全ての単純変数はクリアされ，配列変数は全て抹消されます．

3.2.8 複数条件による分岐(ON GOTO…，ON GOSUB…)

プログラムの中で，場合分けを行うことがしばしばあります．普通，場合分けは IF 文を使って行いますが，数の大きさによって場合分けを行う時に便利な命令があります．それらは ON GOTO 文と ON GOSUB 文です．まずプログラムを見て下さい．

```
100 INPUT N
110 SG=SGN(N)+2
120 ON SG GOTO *MINUS,*ZERO,*PLUS
130 *MINUS
140 PRINT "minus" : GOTO *EXIT
150 *ZERO
160 PRINT "zero" : GOTO *EXIT
170 *PLUS
180 PRINT "plus"
190 *EXIT : END
```

110行のSGNという関数は符号を調べる働きをし，Nが正なら1を，0なら0を，負なら−1を値とします．120行にあるのがON GOTO文で，ONに続く式の値を調べて，その値が1ならGOTO文に続く行番号の並び（ここではラベルが使われています）の1番目の行に分岐します．もし値が2なら2番目，3なら3番目の行に分岐するのです．また値が行番号の数より大きいかまたは0であるなら，次の行に実行が移ります．しかし式の値が負になった場合はエラーとなります．

ON GOSUB文もON GOTO文と動作は同じです．ただ分岐先がサブルーチンになっているだけです．

3.2.9 プログラムの一時停止

プログラムを終了させるには，END文を使いましたが，プログラムの実行を停止させるには，STOP文を使います．STOP文が実行されると，BASICはプログラムがどこで停止したというメッセージを表示してコマンドレベルになります．

```
new
Ok
100 print "start"
110 stop
120 print "continue"
130 end
run
start
Break in 110
Ok
■
```

この状態でCONT命令を入力すれば，プログラムの実行を再開することができます．このCONT命令は，STOPキーでプログラムの実行を中断した場合も，その実行を再開することができます．

このSTOP文は，プログラムが正常に働いているかどうかを確かめる時によく用いられます．プログラムの中に幾つかSTOP文をおいておくと，プログラムの実行がそこに来ると停止しますから，プログラムの実際の流れや，プログラムの中で使っている変数の内容を直接モードで確かめることができる訳です．

3.2.10 変数の型宣言

変数の型を定めるには，その変数に型宣言文字を付けることによって行いました．単精度型の

変数以外の変数を使いたい場合は，いちいち変数に型宣言文字を付ける必要があります．

ところで，プログラム中で使用している変数に型宣言文字が省かれている場合に，単精度型ではなく他の型の変数とみなしてくれると便利なことがあります．例えばプログラム中で整数型の変数しか使わない場合などで，全部の変数に％を付けるのは大変面倒です．このような場合のために，通常（型宣言文字が付いていない状態）の変数の型を定義する命令が用意されています．それらは次の４つです．

DEFINT	（整数型）	を定義します．
DEFSNG	（単精度型）	
DEFDBL	（倍精度型）	
DEFSTR	（文字型）	

これらの命令は変数名の頭文字を指定して，その文字で始まる変数の型を定義する働きをします．例えば次のように使います．

```
100 DEFINT I-L
110 DEFDBL B,D
120 DEFSTR C,R-T
      .
      .
      .
```

100行でＩ，Ｊ，Ｋ，Ｌの文字で始まる変数を整数型に定義しています．110行ではＢ，Ｄで始まる変数を倍精度型に，120行ではＣ，Ｒ，Ｓ，Ｔで始まる変数を文字型に定義しているのです．なお定義されなかった文字で始まる変数は単精度のままです．

これらの宣言を行わなかった場合には，最初から BASIC によって DEFSNG　Ａ－Ｚが実行されていると考えることができます．またこれらの型宣言よりも型宣言文字が常に優先します．例えば，プログラム中で整数しか扱わないと考えて全部の変数を整数型に宣言してしまっても，一部で文字型などを使いたい場合には，その型宣言文字を変数に付けて使えばよい訳です．

```
100 defint a-z
110 i=3.14 : i!=3.14
120 i$="string"
130 print i,i!,i$
140 end
run
 3             3.14          string
Ok
■
```

3.3 関数は道具として

N_{88}-BASICには数多くの関数が用意されています．関数とは，ある値を引数として与えると，その値に対してある決まった演算または操作を行うものです．そして，その演算及び操作の結果

を関数の値として返します.

関数は次の形式で使います.

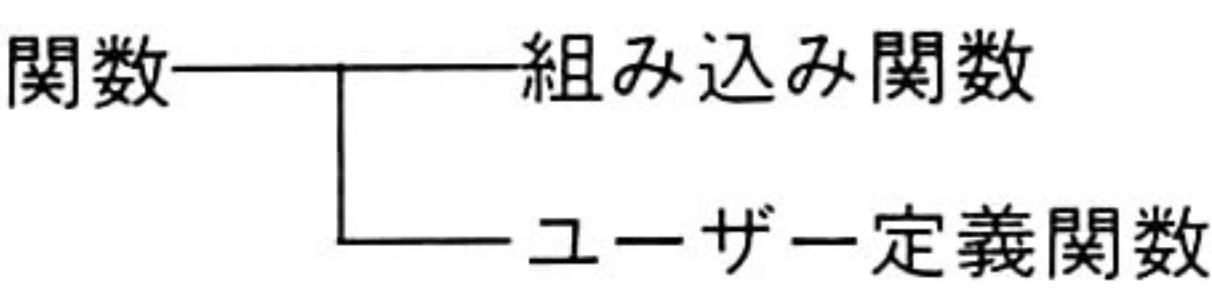
関数名（引数）

引数で表される値に，関数名で示される演算を行って，その結果を与えるのです.

BASIC の関数は代入文の右辺，PRINT 文の中などに使用することができますが，それ単独で記述したり，代入文の左辺などに用いることはできません.

正しい例：A＝SIN（3.14）　　PRINT　CHR$（65）
悪い例　：COS（0）　　　　　CHR$（66）＝〝B〞

関数は，組み込み関数とユーザ定義関数に分けることができます．組み込み関数は予め BASIC に内蔵されている関数で，ユーザー定義関数は，私たちが自由に定義することのできる関数です.

```
関数─┬─組み込み関数
     └─ユーザー定義関数
```

3.3.1 組み込み関数

☆数値関数

数値関数は値が数値で得られる関数です．BASIC には平方根や三角関数などの初等関数や，絶対値を求めたり数値の変換を行う関数などが用意されています.

またN_{88}-DISK BASIC の初等関数では，引数が倍精度の時は値も倍精度で得られます．その他の場合では値は単精度です．単精度の有効桁数は 7 桁，表示は 6 桁ですが，倍精度の場合は16桁です．より高い精度で初等関数を用いた計算ができるようになります.

SQR

平方根を求める関数です．引数は 0 または正の値でなければなりません.

```
print sqr(2),sqr(2#)
 1.41421        1.414213562373095
Ok
■
```

ABS

絶対値を求める関数です.

```
print abs(-2), abs(-0.2)
 2              .2
Ok
print abs(-1.23E21),abs(1.123D21)
 1.23E+21       1.123D+21
Ok
■
```

SGN

引数の符号を調べる関数です．引数の値が正の場合は1を，0の場合は0を，負の場合は－1を値とします．

```
print sgn(-3), sgn(0), sgn(0.2)
-1             0               1
Ok
■
```

INT

引数の値を超えない整数値を求めます．

```
print int(3.1),int(2.7),int(-2.7)
 3             2             -3
Ok
print int(1234567.8!),int(1234567.8#)
 1.23457E+06   1234567
Ok
■
```

FIX

引数の小数点以下を取り除いた値を求めます．

```
print fix(3.1),fix(2.7),fix(-2.7)
 3             2             -2
Ok
print fix(1234567.8!),fix(1234567.8#)
 1.23457E+06   1234567
Ok
■
```

CINT

引数の小数点以下を四捨五入して整数に変換します．

```
print cint(3.1),cint(2.7),cint(-2.7)
 3             3             -3
Ok
■
```

INT，FIX，CINT の働きの違いをよく理解して下さい(INT，FIX の結果はあくまで実数型，CINT は整数型という違いもあります)．

```
100 print "  n            int(n)          fix(n)          cint(n)"
110 for i=1 to 6
120  read n
130  print n,int(n),fix(n),cint(n)
140 next i
150 end
160 data -2.1,-1.6,-0.3,0.3,0.6,1.2
```

```
run
  n              int(n)          fix(n)          cint(n)
-2.1              -3              -2               -2
-1.6              -2              -1               -2
-.3               -1               0                0
 .3                0               0                0
 .6                0               0                1
 1.2               1               1                1
Ok
■
```

CSNG

引数を単精度型実数に変換します.

```
print csng(12300%), csng(1234567#)
 12300          1.23457E+06
Ok
■
```

CDBL

引数を倍精度型実数に変換します. ただし有効数字の桁数は変化しません.

```
print cdbl(12300%), cdbl(1.23E21)
 12300          1.230000190016828D+21
Ok
■
```

これら CINT, CSNG, CDBL 関数は, 数値の型を変換する型変換関数ですが, 実際にはこれらの関数を使わなくても代入文においては型の変換が自動的に行われます.

```
100 a!=1234.56789#
110 b!=csng(1234.56789#)
120 print a!,b!
130 a#=1234.567!
140 b#=cdbl(1234.567!)
150 print a#,b#
160 end
run
 1234.57        1234.57
 1234.567016601563              1234.567016601563
Ok
■
```

ですから代入文においてこれらの関数を使うことは, あまり意味がありません. 代入の際に行われる型の変換を, はっきり示すために用いるぐらいです. また PRINT 文において型変換を行って値を表示するために使うことができます.

LOG

自然対数（e を底とする対数）を求める関数です. 引数の値は正の数でなければなりません.

```
print log(10), log(10#)
 2.30259        2.302585092994046
Ok
■
```

EXP

引数の指数関数（e^x：xは引数）を求めます.

```
print exp(1), exp(1#)
 2.71828        2.718281828459045
Ok
■
```

SIN

引数の正弦（sinx：xは引数）を求める関数です. 引数の単位はラジアン（rad）です.

```
print sin(1.5708), sin(1.5708#)
 1              .9999999999932538
Ok
■
```

角度［deg］の単位をラジアンに変換するには，次の式を使うとよいでしょう.（R：ラジアン，D：度）

R＝D＊（3.141593／180）

COS

引数の余弦（cosx：xは引数）を求める関数です.

```
print cos(3.1416), cos(3.1416#)
-1             -.9999999999730152
Ok
■
```

TAN

引数の正接（tanx：xは引数）を求める関数です. 与える値はラジアンであり，数の値が$\frac{\pi}{2}$の倍数の時は，オーバーフローとなります.

```
print tan(0.7854), tan(0.7854#)
 1              1.000003673321185
Ok
■
```

ATN

引数の逆正接（$\tan^{-1}x$：xは引数）を求める関数です. 結果は$-\frac{\pi}{2}$から$\frac{\pi}{2}$までのラジアンで求まります. ただし，ATNは，N_{88}-DISK BASICでのみ使うことができます.

```
print atn(1), atn(1#)
 .785398        .7853981633974483
Ok
print atn(1)*4,atn(1#)*4
 3.14159        3.141592653589793
Ok
■
```

RND

RND は乱数を与える関数です．乱数の値は 0 以上 1 未満の単精度実数です．

RND 関数で得られる乱数は，引数の値によって次のように異ります．

・引数が正の値―乱数を発生します．RND 関数は引数を省略して（RND の形で）使うと，引数が正の値の場合と同じになります．

・引数が 0 の値― 1 つ前に発生した乱数の値を与えます．

・引数が負の値―新しい乱数系列を定め，乱数を発生させます．

```
new
Ok
100 for i=1 to 8
110  print rnd
120 next i
130 end
run
 .245121
 .305003
 .311866
 .515163
 .0583136
 .788891
 .497102
 .363751
Ok
■
```

乱数系列とは何でしょうか．BASIC は乱数を発生させるために，人間にとってはランダムに見えるほどの周期の長い数列を作って，その数列から数を取り出して乱数としているのです．この乱数の源となる数列が乱数系列という訳です．この関数は，ゲーム，シミュレーションなど使いかってが多いものです．

乱数系列を変えるには，RANDOMIZE 文を使うこともできます．RANDOMIZE 文で整数値を指定すると，その値を使って乱数系列を変更します．もし値を省略すれば，メッセージを表示してその値の入力を求めてきます．

```
100 randomize
110 for i=1 to 8
120  print rnd
130 next i
140 end
run
Random number seed (-32768 to 32767)? 123
 .919946
 .261188
 .772272
 .0849298
 .631966
 .409141
 .880976
 .471944
Ok
■
```

プログラムを何回か実行させて，異なる乱数の種を入力してみると，乱数の出方が変わることが分かるでしょう．また同じ値を入力すると，発生する乱数が同じになることも分かると思います．

乱数系列はプログラムが実行されるといつも同じに設定されます．そのためRANDOMIZE文でその都度乱数系列を変えないと，乱数を使ったプログラムはいつも実行結果が同じというようなことが起こることになります．

今までの説明から分かるように，BASICで発生される乱数は完全にランダム（？）という訳ではありません．乱数らしきものが使えるといった程度に考えて下さい．

SEARCH

整数型の配列変数の要素の中から，任意の値を捜し出し，その要素ナンバーを返す関数です．この関数は，チョット複雑ですから，次にその書式を示しておきます．

SEARCH（配列変数名，捜し出す値〔，捜し始める要素ナンバー，ステップ値〕）

例　A＝SEARCH（B％，50，10，4）

上の例では，配列B％の要素の中から，50という値を捜しています．この時，捜し始める要素ナンバーは10であり，4ステップ（間3つ飛びこす）で捜します．

この関数は，5章で扱うランダムアクセスファイルのインデックスのサーチなどに使うことができるでしょう．但し，この関数はN_{88}−DISK BASICのみ．

☆文字関数

文字関数は結果が文字型の値で得られたり，文字データを引数として用いるような関数です．キャラクタコードを扱うものや，文字列の演算を行うものがあります．

キャラクタコード

BASICは文字型のデータを扱うことができますが，これらのデータはすべてキャラクタコードと呼ばれる番号を使って処理されているのです．キャラクタコードとはBASICが扱うことのできる全ての種類の文字に付けられた番号です．番号は0から255までの整数で，16進で表すと&H00から&HFFまでの値です．

BASICで扱える全ての文字と，そのキャラクタコードとの対応を表したのが次の表です．

最初の横に並んだ0～Fの16文字は，キャラクタコードの上位4ビットを表す16進数で，表の左の縦に並んだ0～Fの文字は，キャラクタコードの下位4ビットを表す16進数です．この表を見ると英字のAのキャラクタコードは&H41（10進で65）と分かります．

ところで&H01～&H1Fのキャラクタコードのところには見慣れない記号がありますが，これはコントロールキャラクタの略号です．実際にはこのような記号は表示されません．

上位4ビット→
下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		
1	S_H	D_I	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	C_R	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N		n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	_	o				ッ	ソ	マ	゜		

表1　キャラクタコード表

CHR$

引数をキャラクタコードとして，その文字を与えます．引数の値はキャラクタコードですから，0～255の範囲になければなりません．

```
100 print chr$(65)
110 a$=chr$(65)+"BC"
120 print a$
130 a$=chr$(34)+a$+chr$(34)
140 print a$
150 end
run
A
ABC
"ABC"
Ok
■
```

Aのキャラクタコードは65ですから，CHR$(65)と〝A〞は全く同じものです．また引用符のキャラクタコードは34ですから，CHR$(34)を使えば文字変数に引用符を代入することもできます．

また CHR$ 関数を使えば，コントロールキャラクタを取り扱うことができます．

ASC

引数の文字のキャラクタコードを与えます．引数の文字列が 2 字以上の時は，最初の文字のキャラクタコードを与えます．

```
new
Ok
100 print asc("A")
110 print asc("ABC")
120 print asc(chr$(34))
130 end
run
 65
 65
 34
Ok
■
```

引数の文字列がヌルストリングの場合はエラーとなります．

ASC 関数は，CHR$関数の逆関数です．

☆LEN—文字列の長さ

引数の文字列の長さを与える関数です．文字列の長さとは，その文字列を構成している文字の数です．もし文字列の中に空白やコントロールキャラクタが含まれていたら，その文字も数えます．

BASIC では最大255個の文字を 1 つの文字列として扱うことができます．最小は 0 個で，これはヌルストリングです．LEN 関数の結果は 0 ～255の値となります．

```
100 c$="beep"
110 print c$,len(c$)
120 c$=c$+chr$(7)
130 print c$,len(c$)
140 end
run
beep           4
beep           5
Ok
■
```

☆RIGHT$, LEFT$, MID$

これらは，文字列の一部分を取り出す働きをする関数です．

RIGHT$(X$, N)

文字列 X$ の右からN個の文字を取り出します．Nが X$ の長さを越えた時は，X$ 全部を値とします．Nが 0 の時は，結果はヌルストリングとなります．Nは 0 ～255の範囲になければなりません．

```
100 x$="PC-8801"
110 print right$(x$,4);"/"
120 print right$(x$,10);"/"
130 print right$(x$,0);"/"
140 end
run
8801/
PC-8801/
/
Ok
■
```

RIGHT$ 関数は，文字列の右詰め出力に利用されます．

LEFT$(X$, N)

文字列 X$ の左，つまり先頭から，N文字だけ取り出します．その他の機能は RIGHT$ と同じです．

```
100 x$="PC-8801"
110 print left$(x$,2);"/"
120 print left$(x$,10);"/"
130 print left$(x$,0);"/"
140 end
run
PC/
PC-8801/
/
Ok
■
```

LEFT$ 関数は，文字列の冗長部分を切り捨てたり，左詰めを連続して行うのに使われます．

MID$(X$, M, N)

文字列 X$ の先頭よりM番目の文字から後のN個の文字を取り出します．Mは 1 ～255，Nは 0 ～255の範囲の値でなければなりません．

```
100 x$="PC-8801"
110 print mid$(x$,5,2);"/"
120 print mid$(x$,5,9);"/"
130 print mid$(x$,15,4);"/"
140 end
run
80/
801/
/
Ok
■
```

M番目の文字以後の文字列の字数がN以下の場合は，その文字列全部が結果となります．またMが X$ の長さよりも大きい場合は，結果は必ずヌルストリングとなります．

MID$ の 3 番目の引数Nは省略することができます．こうするとNの値に 255を用いた場合と同じで，X$ のM番目以後の文字全部を結果とします．

```
new
Ok
100 x$="PC-8801"
110 print mid$(x$,4);"/"
120 end
run
8801/
Ok
■
```

STRING$

指定された文字が並ぶ文字列を与える関数です．使い方は次の2通りがあります．

STRING$(N, C)

STRING$(N, X$)

前者は，Cの値をキャラクタコードとする文字をN個並べた文字列が得られます．後者はX$の先頭の文字をN個並べたものが得られます．

```
100 c$=string$(10,65)
110 print c$
120 c$=string$(10,"ABC")
130 print c$
140 end
run
AAAAAAAAAA
AAAAAAAAAA
Ok
■
```

NとCの値は両方とも0～255の範囲になければなりません（Cはキャラクタコード）．Nの値が0の場合は，文字列はヌルストリングとなります．

SPACE$

引数の値だけの空白を持った文字列を与えます．引数の値は0～255の範囲になければなりません．

```
new
Ok
100 c$="ABC"+space$(6)+"JKL"
110 print c$
120 end
run
ABC      JKL
Ok
■
```

SPACE$(N)はSTRING$(N, 32)と同じ働きです（キャラクタコード32＝スペース）．

SPC, TAB

どちらもPRINT文の中で用いて，空白を出力する働きがあります．他の関数とはチョット異なり，PRINT文中以外では使用できません．

SPC(N)

Nの数だけ空白を出力します．Nの値は0～255の範囲になければなりません．

```
100 print "1234567890123456789"
110 print spc(7);"abc";spc(3);"ghi"
120 end
run
1234567890123456789
       abc   ghi
Ok
■
```

TAB(N)

画面上の行において，現在のカーソルの位置から，指定した桁位置まで空白を出力します．Nの値は画面の横方向の位置（Nは0から）を指定するのです．

```
100 print "1234567890123456789"
110 c$="8801"
120 print tab(10);c$
130 print c$;tab(6);c$
140 print tab(90);c$
150 print tab(6);c$;tab(9);c$
160 end
run
1234567890123456789
          8801
8801  8801
          8801
      8801
         8801
Ok
■
```

引数の値が現在の横表示文字数（80または40）を超えている場合は，表示文字数とMODをとり，その値のTABを出力します．また，引数の値が現在のカーソルの位置より小さい場合は，改行が行われて表示は次の行で行われます．

SPCとTABはどちらも画面の表示を目的で使われるものです．SPCは空白の数を指定し，TABは次の表示の位置を指定することに注意して下さい．

STR$とVAL

今まで数値や文字列のデータをいろいろ扱ってきました．数値型と文字型のデータはどこに違いがあるのかも理解されたのではないでしょうか．

それでは例を見て，数値と数値を表す文字列の違いを再認識して下さい．

```
100 a=123.45 : a$="123.45"
110 print a
120 print a$
130 end
run
 123.45
123.45
Ok
■
```

数値型のデータの出力は，BASICが数値を文字列に変換して，書式を整えて行われます．それに対して文字型のデータは，それが何であろうとそのまま出力されるのです．

数値と文字列は見かけが同じ場合でも，内部では全く別なものとして扱われています．ところがプログラムを作る上で，文字列で表現されている数字を実際の数値として取り扱いたい場合があります．また，その逆の場合もしばしば起こります．そのような時，文字と数値を互いに変換することができるのが，STR$ と VAL の2つの関数です．

STR$(X)

数値を文字列に変換する関数です．数式Xの値を，それが表示された場合と同じ文字列に変換します．

```
a$=123.45
Type mismatch
Ok
a$=str$(123.45)
Ok
print a$
 123.45
Ok
■
```

数値はそのままでは文字変数に代入できませんが，文字に変換すれば代入することができます．数値を文字に変換すれば，文字列と同じように演算を行うことができるのです．

例として数値を文字に変換して右詰めで表示させたものを示します．

```
100 for i=1 to 5
110  read a
120  print right$("     "+str$(a),6)
130 next i
140 end
150 data 1,12,123,1234,12345
run
     1
    12
   123
  1234
 12345
Ok
■
```

VAL (X$)

数値を表現する文字列 X$ を数値に変換します．X$ の先頭の文字が数値を表すものでなければ，値は0となります．また，X$ の中に数値を表す文字以外のものが現れると，それ以後の文字列は無効となります．ただし，文字列中の空白は無視します．

```
a="123.45"
Type mismatch
Ok
a=val("123.45")
Ok
```

```
print a-3
 120.45
Ok
■
```

INSTR（文字列の検索）

ある文字列中に含まれている文字や文字列を捜し出す働きをする関数です．

INSTR (X$, Y$)

X$ の中から文字列 Y$を捜し，見つかればその位置（X$ の先頭から何番目にあるか）を，見つからなければ 0 を値とします．

```
new
Ok
100 x$="123456789012345"
110 print instr(x$,"789")
120 print instr(x$,"45")
130 end
run
 7
 4
Ok
■
```

INSTR 関数は，文字データを扱うプログラムで文字や語句を検索するために用いられます．しかし上の例でも分かるように，１つの文字列の中に同じ文字が含まれていても，このままでは全部調べることはできません．そこで INSTR 関数には，文字列を捜し始める位置を指定できる機能も備わっています．

INSTR (M, X$, Y$)

X$ の先頭からM番目の文字より Y$ を捜し始めます．Mを必要に応じて捜してやれば，X$ の中の Y$の文字列を全て見つけることができます．

```
100 x$="123456789012345"
110 y$="45"
120 i=instr(x$,y$)
130 print i
140 print instr(i+1,x$,y$)
150 end
run
 4
 14
Ok
■
```

次に示すプログラムは，X$ に含まれる Y$ が幾つあるかを求めるものです．

```
100 input "x$=";x$
110 input "y$=";y$
120 i=0 : n=0
130 i=instr(i+1,x$,y$)
140 if i<>0 then n=n+1 : goto 130
150 print n
160 end
```

```
run
x$=? 12312
y$=? 12
 2
Ok
■
```

HEX$, OCT$

16進，8進を表す文字列を与える関数です．

メモリの内容を直接読み書きしたい時や後ででてくるPAINT文のタイルストリングなどに使います．しかし，8進を表すOCT$は，ほとんど使うことはないでしょう．

```
new
Ok
100 for i=1 to 7
110  read n%
120  print n%,hex$(n%),oct$(n%)
130 next i
140 end
150 data 2,8,16,32,64,128,256
run
 2             2             2
 8             8             10
 16            10            20
 32            20            40
 64            40            100
 128           80            200
 256           100           400
Ok
■
```

☆その他の組み込み関数

N_{88}-BASICはこれまでに紹介した関数の他に，グラフィック用の関数，入出力関数，特殊関数など，様々な用途に合うような，関数が用意されています．これらの関数は，以降の章でそのつど詳しく解説することにします．

3.3.2 ユーザー定義関数

BASICでは組み込み関数の他に，ユーザー（プログラマ）が自由に関数を定義して使うこともできるようになっています．同じ式を何度も使うような場合に，それを関数として定義しておくと，その関数を使えばその式の値がすぐに求まるのです．

関数の定義の仕方を示します．

DEF　FN名前（引数の列）＝定義式

FNと名前をいっしょにしたものが関数の名前です．名前の付け方は変数名の場合と同じです．引数の列は定義式の中で使われる変数に対応しており，仮引数と呼ばれています．

100 DEF FNZ(X, Y)＝SQR(X＾2＋Y＾2)

```
110 DEF FNA(N)=N/3+5
120 DEF FNRND9=INT(RND*9+1)
```

仮引数は関数が引用された時に，その引数として使われた値を，関数の定義式の変数に引き渡す働きをします．例えば110行で定義されている関数が，FNA(2.5)と使われた場合は，2.5／3＋5という演算が行われて，その結果が FNA (2.5) の値となります．

この関数を定義している式の中で使われた変数は，何の影響も及ぼしません．つまり変数Nがプログラム中で使われている場合に，関数 FNA を使っても，Nの値が変ったりすることはないのです．

120行で定義を行っているように，引数を持たない関数を使うこともできます．また引数の列の中にない変数を定義式の中で使ってもかまいません．この変数の値は，その関数が使われた時にプログラム中にある同じ名前の変数の値が使われます．

関数の名前で表される型は，定義式で得られる値の型と一致していなければなりません．たとえば，定義式で得られる値が文字列ならば，関数名も文字型のものでなければいけません．また関数が使われた時の引数の数や型は，仮引数のそれと一致していなければならないのです．

DEF FN での関数の定義は，式の値を計算するサブルーチンと考えることができるでしょう．しかし引数には，その関数が使われた時に，プログラム中にある同じ名前の変数の値が使われます．

```
100 def fnxy(x)=x+y
110 y=5
120 print fnxy(3)
130 end
run
 8
Ok
■
```

関数の名前で表される型は，定義式で得られる値の型と一致していなければなりません．たとえば定義式で得られる値が文字列ならば，関数名と文字型のもの（たとえば文字型宣言文字＄を付ける）でなければなりません．また関数が使われた時の引数の数や型は，仮引数のそれと一致していなければならないのです．

ユーザー定義関数は，結果の値が数値でも文字でもいい訳です．数値が値として求まる例は前に示しましたから，次に文字列が結果となるユーザー定義関数の例を見てみましょう．前に説明しましたように，結果が文字型になる訳ですから，関数名も文字型のものでなければなりません．文字型のユーザー関数は，後で述べる文字関数を組み合わせて複雑な文字列演算を行う場合などによく用います．

DEF　FN 文での関数の定義は，式の値を計算するサブルーチンと考えることができるでしょう．しかし引数によってデータを引き渡すことができ，結果も変数を介することなく直接引用で

きますから，サブルーチンよりも便利です．必要に応じてサブルーチンと使い分けるとよいでしょう．

```
new
Ok
100 def fna$(x$)="This is a "+x$+"."
110 print fna$("pen")
120 end
run
This is a pen.
Ok
■
```

4章　グラフィックス・ワールド

この章では，N_{88}-BASIC のグラフィックスを使いこなす上での，基本的事柄について，説明していきたいと思います．内容は，基礎的なことを中心にしましたが，サンプルプログラムを多く導入し，各命令・機能に対しての具体的なプログラムで，理解をより確実なものにして頂けたらと思っています．この章の最後に，各命令をマスターしたのちの整理のために，グラフィックに関する命令を一覧表にまとめておきました．座標などで混乱してきたら是非活用して下さい．

本章の実行結果の中で特にカラーを必要とするものは，冒頭のカラーページに掲載されています．

4.1　PC-8801の画面構成

ここでいう画面とは，テキスト画面とグラフィック画面の両方を指しています．これらの2つの画面は，独立しています．分かり易く言えば，一方の画面を操作しても，もう一方の画面は，その操作によって影響を受けないということです．例えば，[HOME CLR]のキーを押した時，それはテキスト画面を消すのみで，グラフィック画面は消えません．このことを利用すると，テキスト画面とグラフィック画面を合成して，美しい表やグラフの作成などに役立てることができます．更に言えば，独立しているということは，各画面それぞれについて，別々にモードを設定する手間がかかります．では，2つの画面について，それらがどのようなものか，また，それぞれのモードの設定の仕方，使い方を紹介しましょう．

4.1.1　テキスト画面

この画面は，キーボードから入力した文字や，プログラムのリストなどを表示するための画面です．この画面に表示する文字数は，縦と横と別々に設定することが可能です．

縦に出せる文字数，つまり行数のことですが，これは，20行と25行のモード（様式）があり，横に出すことのできる文字数，つまり桁数は，40文字(桁)と80文字(桁)のモードがあります．この縦と横の文字数を設定するためには，WIDTH という命令（コマンド）を用います．この命令の使い方は，（表１）に従います．

横 ＼ 縦	80 文 字	40 文 字
20行	WIDTH 80, 20↲	WIDTH 40, 20↲
25行	WIDTH 80, 25↲	WIDTH 40, 25↲

表1 WIDTHの設定

この行数と桁数のどちらか一方だけを変えるときに，行数に限り，現在のままでよいとき省略することができます．例えば，画面が今80文字×20行のモードであったとします．そのとき，もしあなたが横だけを40文字にしたいなら，

```
WIDTH 40
```

で40文字×20行になります．しかし縦だけを25行にしたい時は，横の文字数の省略を許さず，

```
WIDTH 40, 25↲
```

と書かねばなりません．起動時の画面表示数は本体のＤＩＰスイッチで選択することができます．ですから，自分で好きな状態に設定しておくことができます．

更に，注意しておくことがあります．それは，行や桁数の数え方です．今，80文字×25文字のモードだとすると，行数は25行ありますが，これは第０行から第24行までの25行ということです．要するに画面で見る１行目は，第０行のことなのです．第何行か，ということを指定するときには，０行１行２行……と数えていくように心掛けて下さい．

もう一つ，テキスト画面を操作する命令として，CONSOLE があります．この命令は最高４つの値が必要で，これをパラメータ（引数）と呼びます．この４つのパラメータで設定することは，次の３つのことです．

① スクロール・ウィンドウ（スクロールする領域）について
② ファンクション・キー表示について
③ 白黒／カラーについて

これらを実際に CONSOLE 文で，どのように値を指定するかというと，

CONSOLE 〈スクロール開始行〉，〈スクロール行数〉，〈②について〉，〈③について〉

と書けば（もちろんキーボードから入力する訳です）よいのです．

①のスクロールする領域の決定のためには，２つの値（パラメータ）を与える必要があります．

```
CONSOLE 10, 5↲
```

と書いたとしましょう．これは，『第10行目から５行分の間で，スクロールする』と解釈されます．このときの第何行目という値の最大値は，WIDTH 文での行数の指定によって異なります．設定

したあとに数十行のプログラムのリストをとってみて下さい．第10行目から5行の間だけで，プログラム・リストが現れるはずです．

次に画面の下に出ている，白い四角形の中に文字の書いてある5つのもの（ファンクション・キー）が，「ちょっと，邪魔だなあ」とか，「グラフィックの絵と重ねたくないなぁ」という人のために，②の機能があります．CONSOLE文の〈②について〉の値と，ファンクション・キー表示についての関係は，（表2）のようになります．

値	ファンクションキー
0	表示されない
1	表示される

表2　ファンクションキー表示スイッチのパラメータ

CONSOLE文もWIDTH文と同様に，現在のままでよい値は省略することができます．しかし，何番目のパラメータであるかを明確にするため，コンマを省略することはできません．例えば，ファンクション・キーが，表示されている状態で，

```
CONSOLE ,, 0 ↵
```

とすると，画面の下からはファンクション・キーが消えます．

```
CONSOLE ,, 1 ↵
```

とすれば元に戻ります．

CONSOLE文の最後のパラメータは，白黒／カラーのそれぞれのモードへ変更するスイッチのようなものです．これを変更するということは，後に述べるCOLOR文における，ファンクション・コードの持つ意味までも変えるということです．今の時点では，

```
CONSOLE ,,, 0 ↵  ：白黒モードにする．
CONSOLE ,,, 1 ↵  ：カラーモードにする．
```

ということだけ，頭の片隅に留めておいて下さい．

以上の4つのパラメータは，初期状態において，

```
CONSOLE 0, 20, 1, 0 ↵
```

を行ったのと同様な状態――すなわち『第0行目から20行の間スクロールし，ファンクション・キーを表示し，白黒モードである』といった状態に設定されています．ここでいう白黒モード，カラーモードというのは，当然テキスト画面だけに影響するものです．次に説明するグラフィック画面とは，完全に独立していることを忘れないで下さい．

4.1.2 グラフィック画面

この画面は，N_{88}の持つ様々なグラフィック命令によって，絵や図を描いたり，消したりといった操作の影響を受ける画面のことです．まず，この画面について話を進める前に，ドット（dot）ということについて少々ふれておきます．

英語の辞書で，ドットという単語を引いてみて下さい．たぶん『点』という意味があると思います．まさしくドットとは，画面上の点のことなのです．グラフィック画面はこのドットの集まりでできています．但し，ここで断っておきたいのは，そのドットを見えるようにするかしないかとか，見えたのなら何色にするか，などを決定するのは読者自身（プログラマ次第）なのです．とかく点というと，目に見えるものしか思いつかない人が多いので，そういった人たちの既成概念を取り除くという意味で，少々説明しておきました．

さて、グラフィック画面というのは，一体幾つのドットからできているのでしょうか？　これを決定するための命令が，SCREEN です．SCREEN 文にも，CONSOLE 文と同じように，4つのパラメータがあります．ここではその第１番目のパラメータだけにふれることにして，残りの３つのパラメータは，4.2.2の座標の所で詳しく述べることにしましょう．

グラフィック画面のドット数ですが（横のドット数）×（縦のドット数）という書き方をすれば，640×200と640×400というモードがあります．更にカラーと白黒のことを考えると，

カラーの640×200
白黒の　640×200
白黒の　640×400（カラーでは，存在しない）

の３つのモードがあります．それぞれのモードの設定は，（表３）のようにして，SCREEN 文で行います．

dot数 ＼ モード	カラー・モード	白黒モード
640×200	SCREEN 0 ↲	SCREEN 1 ↲
640×400	定義されない	SCREEN 2 ↲

表３　グラフィック画面のモード設定

実際にメモリ上でこのグラフィック画面は，640×200の大きさのプレーン（面）と呼ばれるものが３枚（３ページ分）あります．したがって，もう気付いたと思いますが，『白黒モードの640×200』には，３ページ分の画面が用意されています．これらのどのページを表示させるか，どのページに図などを描くか，といった命令は，4.2の座標の所のSCREEN 命令で述べることにします．ここまでくれば，白黒モードの640×400は，３つの640×200のプレーンを２つ，つなげたものであることも想像できるでしょう．いったい，どの２つのプレーンをつなげたのでしょうか？　また，カラーモードの640×200は，この３枚のプレーンをどのように使っているのでしょうか？

これらのことを説明するために，まず３原色の考え方を取り入れましょう．３原色とは，すべ

ての色の基となる3つの色のことです．この3色は普通，青，黄，赤の3色ですが，ここでは光の3原色──青，赤，緑（俗にいう、RGB（Red，Green，Blue）──のことを指します．この3原色と，3つのプレーンが，それぞれに対応しているわけです．プレーン（図1）のような番号をふってみます．

プレーン1	プレーン2	プレーン3
640×200	640×200	640×200
Blue	Red	Green

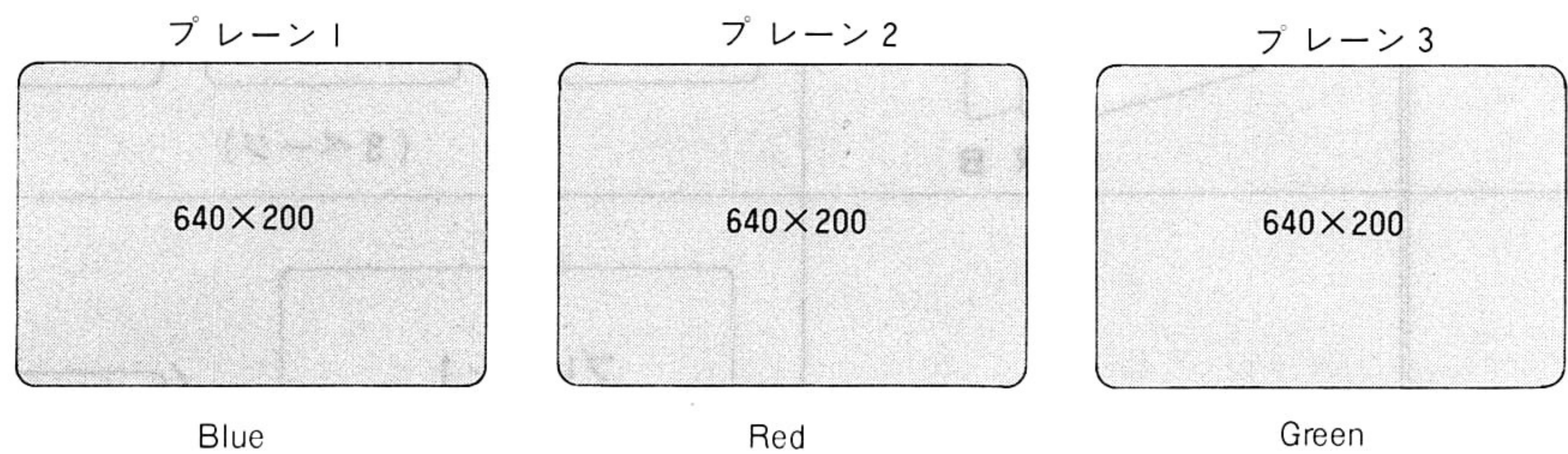



図1　プレーンと色の関係(1)

1つずつ疑問を解決していきましょう．まず，640×400では，いったいどの2つのプレーンをつなげているのか，という疑問です．これは，『プレーン1とプレーン2がつながって，その時プレーン3は，使用していない．』というのが答です．次にカラーモードで，この3枚がどう使われているか，という疑問です．図を見てすぐわかるように，プレーンと（光の）3原色が，それぞれ対応しています．カラーモードでは，この3枚が重ねられて（合成されて）画面に出ているのです．したがって640×400の画面というのは，1枚分のプレーンしかとれませんから，カラーには，なりえないこともわかります．3原色は，その組み合わせによって様々な色が出せるので，その3原色に対応するプレーンの合成も，3原色を組み合わせたのと同じように様々な色が出せる訳です．例えば，青いドットはプレーン1だけにドットが存在し，その他の2つのプレーンには，点がないということです．この3つのプレーンとカラーコードには，（表4）や（図2）の様な関係があります．以上のプレーンのまとめを（表5）に示しておきます．

表4　プレーンと色の関係(2)

プレーンのビットパターンを2進数とみなして10進数にすれば，カラーコードが得られます．

〈色〉	〈プレーン〉 G R B	〈カラーコード〉
黒	0 0 0	0
青	0 0 1	1
赤	0 1 0	2
紫	0 1 1	3
緑	1 0 0	4
水色	1 0 1	5
黄	1 1 0	6
白	1 1 1	7

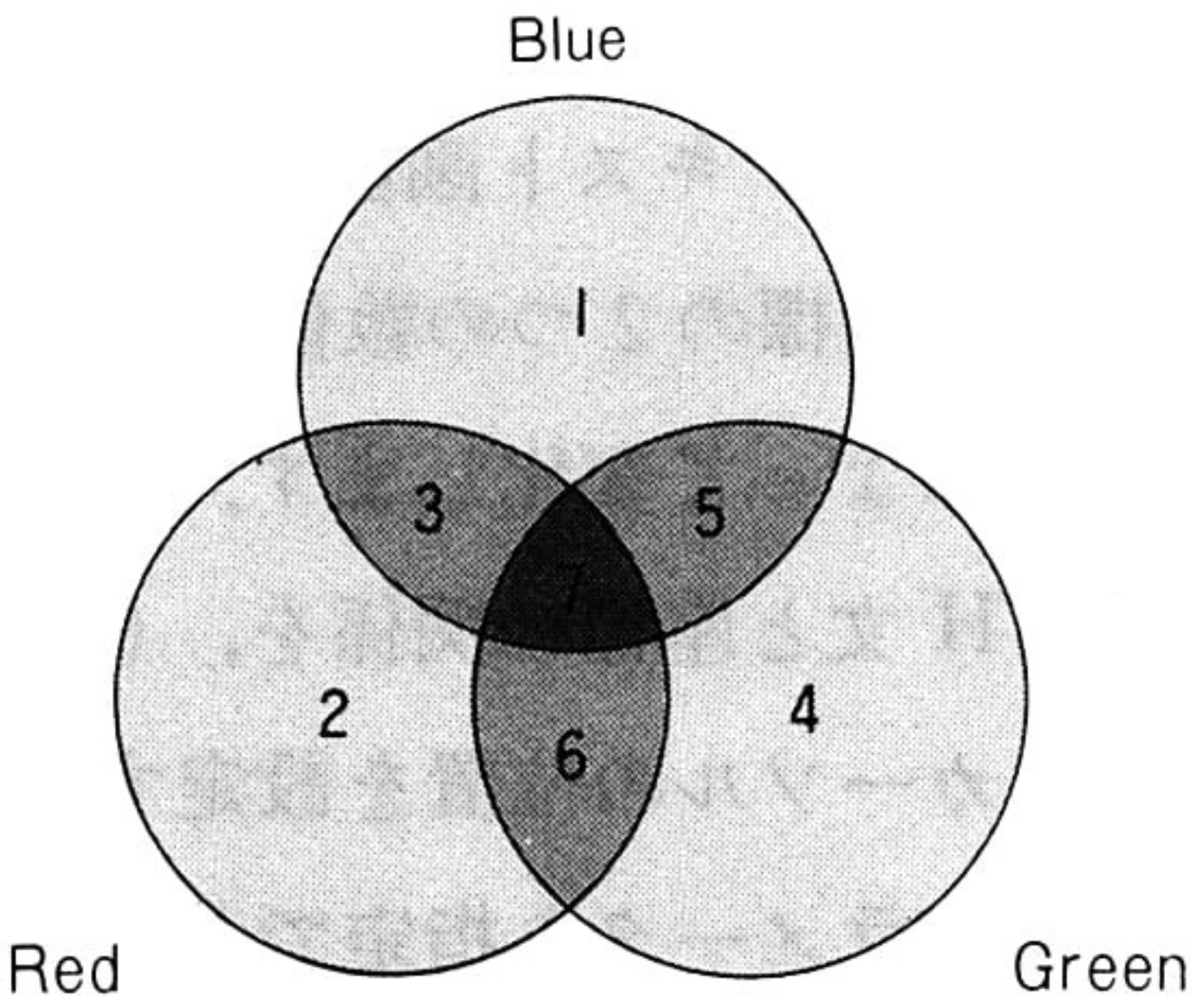



図2　プレーンと色の関係(3)

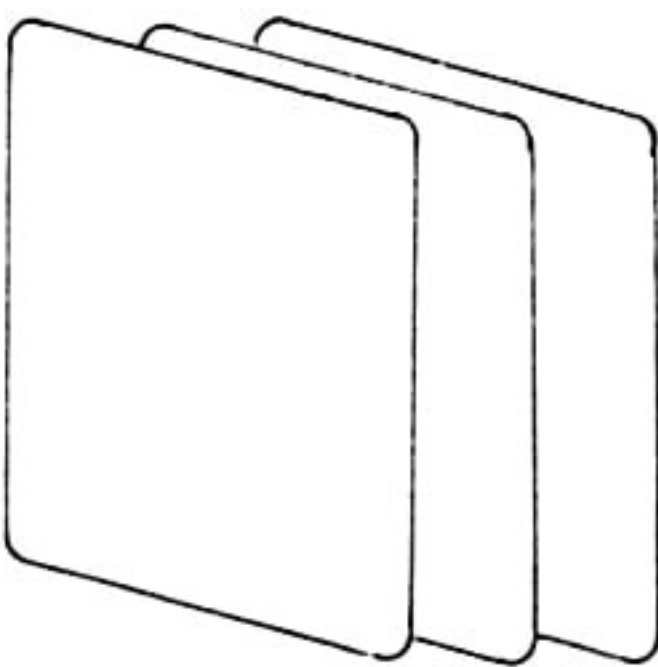
例えば，水色(5)というのは，プレーン1とプレーン3の両方にドットがあることを意味します．

dot数＼モード	カラーモード	白黒モード
640×200	G R B	プレーン1 プレーン2 プレーン3 (3ページ)
640×400	存在しない	プレーン1 プレーン2 (プレーン3 使用されない)

表5　プレーンとモードの関係

4.2　2つの座標

画面上の任意の位置を示すために用いられるのが，座標という考えです．画面にテキスト画面とグラフィック画面があったように，座標もそれぞれの画面に対応して，キャラクタ座標とグラフィック座標があります．

この両者も独立しているという理由で，一方を操作しても，もう一方はそれに影響されず，それぞれの座標について別々にモードを設定しなければなりません．したがって，画面のときと同様に，話を別々に進めた方が理解し易いと思いますので，順に，話をしていきます．

4.2.1　キャラクタ座標

この座標は，4.1.1のテキスト画面上でのみ考えられる座標であり，テキスト画面上での位置を示す水平座標と垂直座標の2つの数値のことです．この数値の最小値は，(0，0)ですが，最大値は，WIDTH 文によって変化します．(0，0)の所を原点と言い，これは必ず画面の左上に位置します．WIDTH 文と座標の関係を，(表1) と (表6) を見比べて，確認しておいて下さい．

座標を用いて，カーソルの位置を設定する命令としてLOCATEが用意されています．この命令は，最高3つのパラメータを指定できます．一般にLOCATEは，

LOCATE　〈横の座標〉，〈縦の座標〉，〈カーソル表示について〉

という文で表現されます．今までの命令と同様に現在のままでよいパラメータは，省略すること

縦＼横	80文字	40文字
20行	(0,0) → (79,0) ↓ (0,19)　(79,19)	(0,0) → (39,0) ↓ (0,19)　(39,19)
25行	(0,0) → (79,0) ↓ (0,24)　(79,24)	(0,0) → (39,0) ↓ (0,24)　(39,24)

表6　WIDTH設定による，キャラクタ座標系

ができます．実際に画面のどの部分に指定した座標があるのか，試してみたい人はプログラム1を入力してRUNさせてみて下さい．但し，プログラムを終了するときは，STOPを使用して，終えて下さい．

```
100 '
110 '   Program 1
120 '     LOCATE demonstration
130 '
140 '
150  CLS : WIDTH 80,25
160  IF X<21 THEN LOCATE 22,0 ELSE LOCATE 0,0
170  INPUT "coordinate (x,y) ";X,Y
180  CLS : LOCATE X,Y
190  PRINT "X"; : GOTO 160
```

カーソルの位置を指定する命令があるのなら，カーソルが今，画面のどこにあるかを知るための命令は，ないのでしょうか？　そういう人たちに応えるためにある関数がPOSとCSRLINです（値を得ることができる命令を，関数と呼びます）．POSは，カーソルの横の座標の位置を与え，一般に，POS（0）と書きます．CSRLINの方は言うに及ばず，カーソルの縦の座標の位置を与えます．この2つの関数と，LOCATEを使って書いて見たのが，プログラム2です．RUNさせると分かりますが，あたかも土の中を掘り進みながら，ときたま現われる餌を，捜し回っているように見えます．各ルーチンは，以下のような働きをしています．

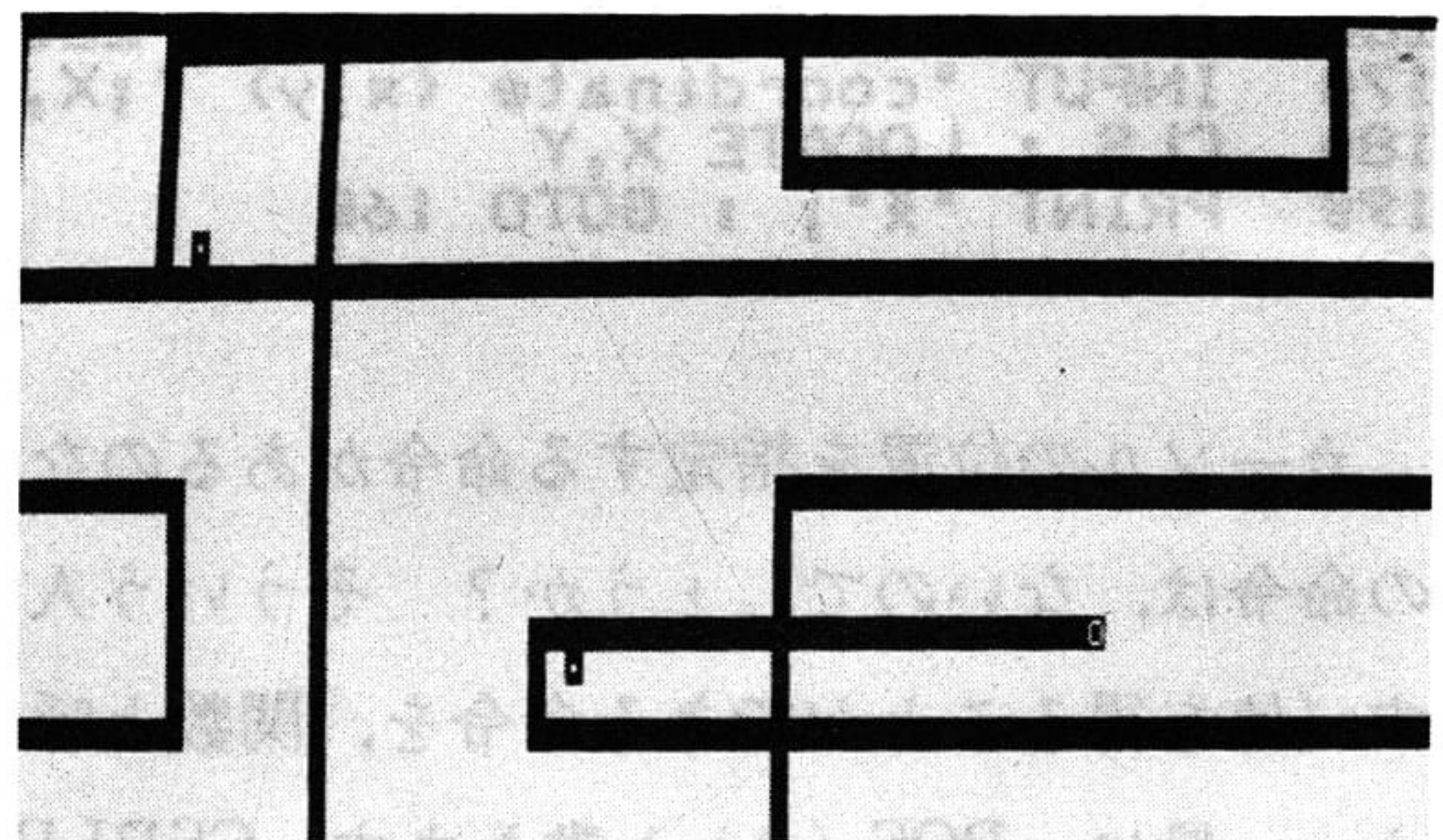
*RIGHTLEFT　：横向きの動きを処理します．

*UPDOWN　：縦向きの動きを処理します．

*XSPACE　：横向きの移動のときに，前に書いた〝○〟または〝－〟を消します．

*YSPACE　　　：縦向きの移動のときに，前に書いた〝0〞または〝1〞を消します．

*OUTCHECK　　：画面からはみ出したときの処理をします．

```
100 '
110 '   Program 2
120 '     LOCATE demonstration
130 '
140 '
150  WIDTH 80,25 : DX=1 : DY=1
160  P=INT(RND(1)*70) : Q=INT(RND(1)*20)
170  X=P+1 : Y=Q+1
180 '
190  FOR I=0 TO 23
200    LOCATE 0,I : PRINT STRING$(80,"■");
210  NEXT I
220 '
230 *RIGHTLEFT
240  GOSUB *XSPACE
250  IF POS(0)=P THEN P=INT(RND(1)*80) : DX=DX*(-1) : GOTO *UPDOWN
260  GOSUB *OUTCHECK
270  LOCATE X,Y : PRINT "o";
280  GOSUB *ESA
290  LOCATE X,Y : PRINT "-";
300  GOTO *RIGHTLEFT
310 '
320 *UPDOWN
330  GOSUB *YSPACE
340  IF CSRLIN=Q THEN Q=INT(RND(1)*24) : DY=DY*(-1) : GOTO *RIGHTLEFT
350  GOSUB *OUTCHECK
360  LOCATE X,Y : PRINT "o";
370  GOSUB *ESA
380  LOCATE X,Y : PRINT "|";
390  GOTO *UPDOWN
400 '
410 *XSPACE
420  GOSUB *OUTCHECK
430  LOCATE X,Y : PRINT " ";
440  X=X+DX
450 RETURN
460 '
470 *YSPACE
480  GOSUB *OUTCHECK
490  LOCATE X,Y : PRINT " ";
500  Y=Y+DY
510 RETURN
520 '
530 *OUTCHECK
540  IF X=-1 THEN X=79
550  IF X=80 THEN X=0
560  IF Y=-1 THEN Y=23
570  IF Y=24 THEN Y=0
580 RETURN
590 '
600 *ESA
610  G=INT(RND(1)*100)
620  IF G<>0 THEN RETURN
630  LOCATE P,Q : PRINT CHR$(&HA5);
640 RETURN
```

プログラムはじっくりと見れば，理解して頂けると思います．

ここで4.1.1の所での CONSOLE 文を思い出してみましょう．4つ目のパラメータは，白黒／カラーの両モード切り換え用になっています．この値によって，COLOR 文におけるファンクション・コードの持つ意味は変わると言いましたが，ここでは，COLOR 文とファンクション・コードの持つ意味をお話ししたいと思います．一般に COLOR 文は，最高 4 つのパラメータで

COLOR ①, ②, ③, ④

と書きます．もちろん変えたくないパラメータや，変える必要のないパラメータは，省略することができます．それぞれのパラメータについて，順を追って説明していきましょう．

まず①のパラメータは，ファンクション・コードです．さっそく出て来ましたが，このパラメータは，値によって，テキスト画面全体の文字に対して様々な働きかけをします．しつこいようですが，ここに置かれる値は白黒モードの時と，カラーモードの時では，同じ値でも，持つ意味が違います．これは長い文で説明するよりも，(表7)を活用した方がよいと思います．要するに，ファンクション・コードは，白黒モードでは，文字の表示の仕方を指定し，カラーモードでは文字の色を指定するものなのです．

ファンクションコード	白黒モード CONSOLE,,, 0	カラーモード CONSOLE,,, 1
0	通常の表示	黒
1	文字は表示されない	青
2	点滅する	赤
3	1と同じ	紫
4	反転した文字を表示する	緑
5	反転して文字を表示しない	水　色
6	反転して点滅する	黄　色
7	5と同じ	白

表7　各モードにおけるファンクション・コードと表示される文字との関係（カラーモードにおけるファンクション・コードのことを特にカラーコードという）

次の②のパラメータは，バック・グラウンド・カラーと呼ばれるパラメータです．これはグラフィック画面における地の色を指定します．色というくらいですから，今まで説明してきた中で，色に関係するような値を指定すればよいということは，察しがつくでしょう．今の段階では，カラーコード（すなわち，カラーモードにおける，ファンクション・コード）を指定すればよいというぐらいに考えておいて下さい．実は，この値は，後に述べるパレットという概念のパレット番号なのですが，これが普通の状態では，カラーコードと一致した値を持っているのでここでは，問題ありません．バック・グラウンド・カラーを指定すると，どのようなとき影響が出るかというと，CLS命令やPRESET命令を実行したときです．各命令については，順を追って説明していきますが，バック・グラウンド・カラーによる影響だけについて述べておきます．CLS命令で，グラフィック画面を消すと，画面は，このバック・グラウンド・カラーで塗りつぶされます．PRESET命令は，色指定なしに実行するとき，バック・グラウンド・カラーで点を打ちます．

③のパラメータは，ボーダー・カラーと呼ばれるパラメータです．ボーダーとは，『境界』という意味があります．理解された方もいると思いますが，ボーダー・カラーとは，境界色のことで

す．境界というのは，この場合，グラフィック画面の境界のことです．（図３）を参照して下さい．これは，パレット番号でなく，カラーコードで指定します．実際，画面のどの部分を指して言っているのかは試してみて自分の目で確認するのが一番だと思うので，是非，やって下さい．（但し，専用高解像度ディスプレイを使っているときには，ボーダー・カラーを指定して色を付けることはできませんから，残念ながら次の命令はエラーとなります．）例えば，

```
COLOR ,, 1 ⏎
```

を実行して，色が青くなった部分の色指定ができるということです．

CRT

グラフィック画面

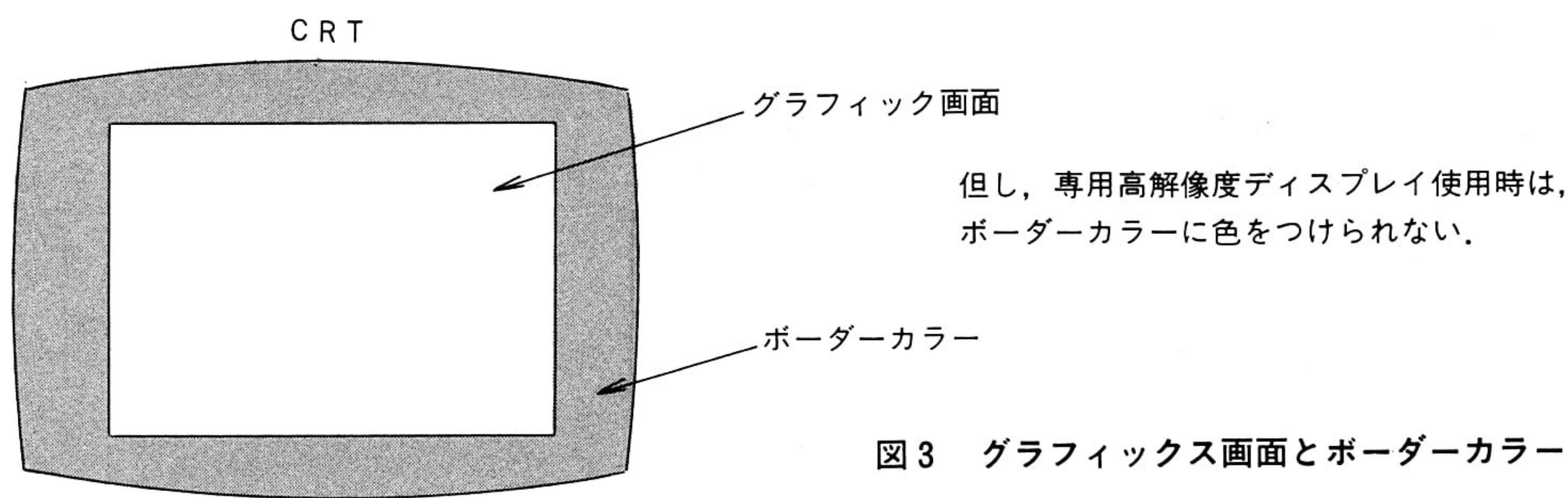
但し，専用高解像度ディスプレイ使用時は，
ボーダーカラーに色をつけられない．

ボーダーカラー

図３　グラフィックス画面とボーダーカラー

最後に④のパラメータです．これは，フォア・グラウンド・カラーと呼ばれ，バック・グラウンド・カラーとは逆の意味を持つのは，すぐに分かります．バックに対して，フォアですから，グラフィック画面の手前の色を指定します．手前の色というのもおかしな言い方ですので，言い換えてみますと，グラフィック画面に，点や線などを描く時に使われる色のことです．フォア・グラウンド・カラーを指定すると，どのような時に影響が出るかというと，N_{88}-BASIC が備えている種々のグラフィック命令を，色指定なしで実行したときです．このパラメータも，バック・グラウンド・カラーと同様にパレット番号によって，その値を示しますが，今の段階では，カラーコードで指定を行っているのと同じと考えても構いません．

ここでCOLOR 文の４つのパラメータの意味と，どんな種類の値を指定すればよいのかを（表８）にまとめておきます．

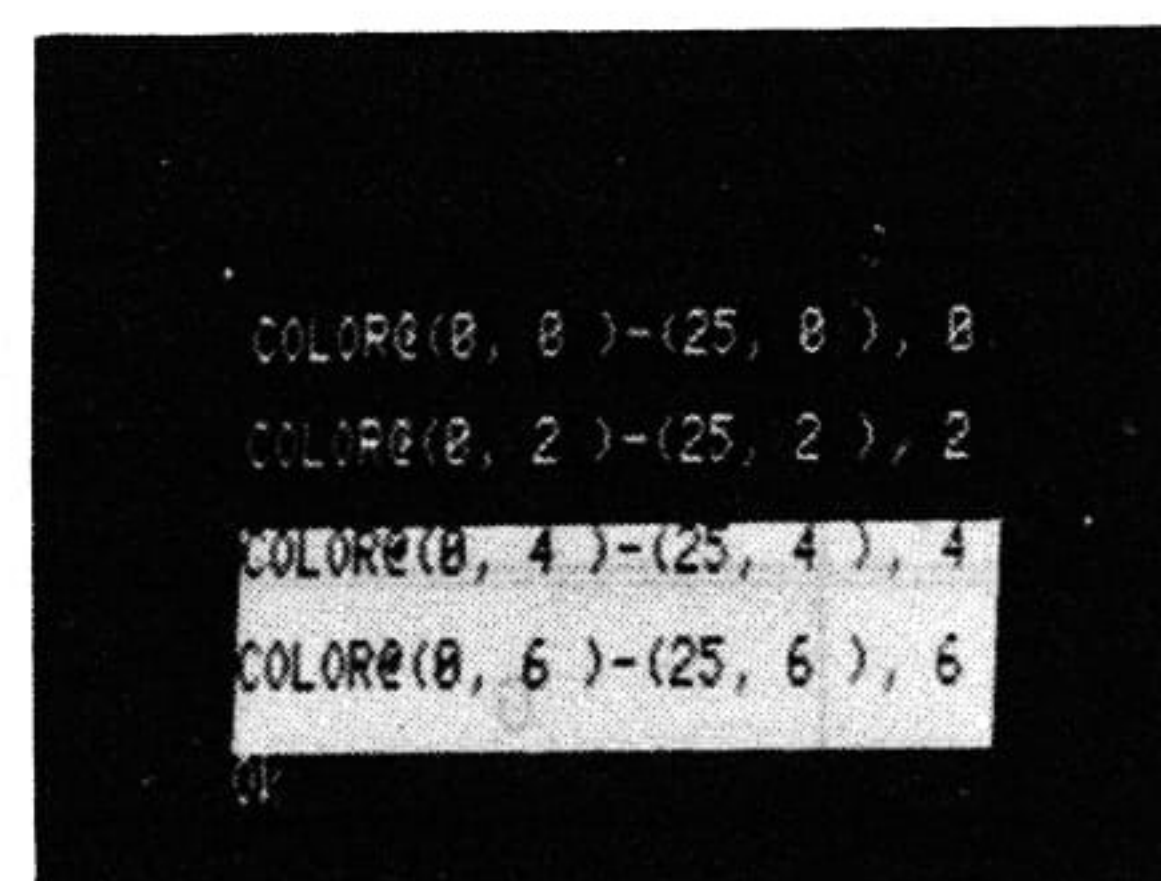
	意　　味	指定する値
パラメータ①	以後，表示する文字に対する機能の設定	ファンクション・コード
パラメータ②	グラフィック画面のバック・グラウンド・カラー	パレット番号
パラメータ③	ディスプレイ画面のボーダー・カラー	カラーコード
パラメータ④	グラフィック画面のフォア・グラウンド・カラー	パレット番号

表８　COLOR文におけるパラメータの意味と指定する値

COLOR 文で，テキスト画面の文字に対して働きかけるパラメータは①だけでした．しかし，この COLOR 文のパラメータ①は，テキスト画面に，これから表示する文字に対して働きかけるものです．では，既にテキスト画面に表示してある文字に対しての機能の変更はできないのでしょうか？　中には，ある一部の文字だけに対して，全体の持っている機能（色）とは別の機能（もしくは色）を持たせたいという人がいるかもしれません．いえ，当然，使っていくうちに必要に感じるでしょう．そういった場合のために用意された命令が，COLOR@です．@は，『単価』という意味がありますから，無理に解釈すれば，『ある任意の一部分』と考えてもいいと思います．この一部分を指定するのに用いられるのが，キャラクタ座標です（COLOR の話が長くて，キャラクタ座標のことを忘れてしまったかもしれませんが，この章の本筋は『キャラクタ座標』ですので，忘れた方は，（表6）の前後を見直して下さい）．

COLOR@(X，Y，)－（X_2，Y_2），〈ファンクション・コード〉

という文で，実行できます．この文は，『キャラクタ座標（X_1，Y_1）と（X_2，Y_2）を対角線とする四角形の範囲に，既に書かれている文字などにファンクション・コードで指定する機能を与える』という命令です．ファンクション・コードは，もちろん（表7）に従います．但し，これが省略された場合，7をその値とします．ここで大事なのは，『既に書かれている文字』ということです．指定した場所に文字がない場合や，指定したあとに書いた文字などには，何の影響もないのです．このことさえ頭の中にあれば，難しいことではないと思います．プログラム3も優しいプログラムですから，RUN させて（表7）と見比べて下さい．

```
100 '
110 '   Program 3
120 '     COLOR@ demonstration
130 '
140 '
150  CLS : WIDTH 80,25
160  INPUT "0) black & white mode. or 1) color mode.";MODE
170  CONSOLE 0,24,,MODE
180  FOR I=0 TO 7
190    L=CSRLIN
200    PRINT "COLOR@(0,";L;")-(25,";L;"),";I
210    COLOR@(0,L)-(25,L),I
220  NEXT
230 END
```

4.2.2　グラフィック座標

この座標は，4.1.2のグラフィック画面上で考えられるものです．グラフィック座標の中には，2つに分けてワールド座標とスクリーン座標という概念があります．この2つの座標系は，少々複雑ですので，今は話を簡単にするためグラフィック座標という1つの座標系しか存在しないものとして，話を進めて行きます．ワールド座標とスクリーン座標は，グラフィックの話に慣れた所で説明したいと思います．

さて，グラフィック座標の本題に入る前に，何故，ワールド座標とスクリーン座標の事について無視するように説明して問題がないのか？　という疑問が生じると思います．これは，初期状態でグラフィック座標とワールド座標，スクリーン座標が全て一致したものとなっているからです．

例えば，任意の点のグラフィック座標が（50，100）であったとしたら，ワールド座標上でも，スクリーン座標上でも，その点の表す座標は，（50，100）であるということです．

では，本題に入って行きたいと思います．キャラクタ座標と同様に，原点（0，0）は，画面の左上の隅になります．またキャラクタ座標がWIDTH文と関係があった様に，グラフィック座標は，SCREEN文と密接な関係があります．その関係は，（表1）と（表3）を見比べれば，すぐ理解できると思います．

4.1.2の所で約束していたので，SCREEN文の4つのパラメータのうちの，残りの3つについて説明しましょう．一般の書式は，次のようなものです．

SCREEN ①，②，③，④

パラメータ①は，前に述べた通りです．（表3）を参照して下さい．

パラメータ②は，画面スイッチと呼ばれるものです．これには，2つの機能があり，ビット指定となっています．ビットというのは，2進数の各桁のことで，情報量の単位です．2つの機能というのは，グラフィック画面を表示させるか，させないかを指定する，グラフィック・マスク・スイッチと，高速に書き込むかどうかを指定する．高速書き込みスイッチです．それぞれのスイッチが，各1ビットに対応しており，2ビットで1つの値として，パラメータの②に与えます．

グラフィック・マスク・スイッチは，ON（1）の状態で，グラフィック画面は表示されません．高速書き込みスイッチは，ON（1）の状態で，高速書き込みモードになります．但し，高速書き込みスイッチが，ONの状態でグラフィック命令を実行すると，画面がちらつくということは覚えておかねばならないでしょう．これらの関係は，（表9）に示しておきました．

グラフィックマスクスイッチ	高速書き込みスイッチ	パラメータ②	機能
0	0	0	普通の速度の書き込みで，グラフィックは表示される．
0	1	1	高速書き込みで，グラフィックは表示される．
1	0	2	普通の速度の書き込みで，グラフィックは表示されない．
1	1	3	高速書き込みで，グラフィックは表示されない．

グラフィックスイッチと高速書き込みスイッチを2ビットの2進数とみなして，10進数に変換するとパラメータ②になる．

表9　SCREEN文のパラメータ②の値とその機能

パラメータ③と④は，パラメータ①が１であるとき，すなわち，640×200の白黒画面が３ページあるときのみ，意味を持つパラメータです．パラメータ③はアクティブ・ページと呼ばれ，グラフィック命令が働きかけるページ，つまり，点や線を描くページを指定します．各ページとは，（図１）のプレーン１から３のことであり，このページをそれぞれに対応した数値，０から２で指定します．

パラメータ④は，ディスプレイ・ページと呼ばれ，３ページのうちのどのページを表示するかを指定します．この指定は，画面スイッチのように，各プレーンがそれぞれ３つのビットに対応しています．ビットを１にすることによって，それに対応したプレーン（ページ）が表示されます．これを用いれば，画面の合成ができます．この３ビットで，表せる値と画面の表示の関係を（表10）に示しておきます．

パラメータ④	表示される画面		
	ページ３	ページ２	ページ１
0			
1			○
2		○	
3		○	○
4	○		
5	○		○
6	○	○	
7	○	○	○

表10　SCREEN文のパラメータ④の値とページ（プレーン）の表示との関係
○印の所を１，何の印もない所を０として，３ビットの２進数とみなすと，パラメータ④の値に一致する．

以上のことで重要なのは，640×200の白黒画面が３ページあるモードでは，必ず自分で利用するページや表示するページを指定して，どのページに描き，どのページを見ているのかということを常に意識することです．そうしないととんでもないページに描いて，画面に何も表示されない．などということが起こり兼ねません．また，SCREEN 文の全体的な注意としては，パラメータ①を指定して実行すると，後に述べる WINDOW や VIEW や LP（最後にグラフィック命令が参照した点の座標）が，初期状態に戻されることです．これらの具体的な状態も，後に述べることにします．

4.3　グラフィック命令

画面や座標については，これまで大体のことを，お話ししました．では，いったいグラフィック

の画面（座標）にはどうやって働きかければよいのでしょう．その働きかけの手段として，N88-BASICが持っている様々な，グラフィック命令を紹介していきたいと思います．その中には，座標の所で話していない，パレットやワールド座標，スクリーン座標なども含まれています．また，話しは，最も良く使うと思われる，640×200のカラーモードを中心に進めていきます．プログラムを参考に，理解して下さい．

4.3.1 CLS, PSET, PRESET—点を描く—

これらの命令は，BASICに用意されている基本的なグラフィック命令の中でも，最も基本的であり，使用頻度の高い命令と言ってもいいでしょう．これから紹介するLINEやCIRCLEなどは，このPSET命令だけで実現することも可能です（ただし，少々数学の知識と時間を必要としますが……）．まずは，プログラム4を見て下さい．

```
100 '
110 '   Program 4
120 '     stardust night
130 '
140 '
150  WIDTH 80,25 : SCREEN 0,0
160  CLS 3
170  FOR I=1 TO 80
180    X=RND(1)*640 : Y=RND(1)*190
190    C=INT(RND(1)*8)
200    PSET(X,Y),C
210  NEXT
220 '
230 '   main routine
240 '
250 *LOOP
260  X=RND(1)*640 : Y=RND(1)*190
270  C=INT(RND(1)*7)+1
280  PSET(X,Y),C
290  SELECT=RND(1)*40+20
300  IF RND(1)<.3 THEN *SKIP
310  IF (C AND 1)=0 THEN GOSUB *RIGHT ELSE GOSUB *LEFT
320 *SKIP
330  PRESET(X,Y)
340 GOTO *LOOP
350 '
360 '   subroutine
370 '
380 *RIGHT
390  L=0
400  *RLOOP : IF SELECT>=78 OR L>=24 THEN *EXIT1
410    LOCATE SELECT,L
420    PRINT "\"; : SELECT=SELECT+1
430    LOCATE SELECT-1,L
440    PRINT " "; : L=L+1
450  GOTO *RLOOP
460 *EXIT1 : RETURN
470 '
480 *LEFT
490  L=0
500  *LLOOP : IF SELECT<=0 OR L>=24 THEN *EXIT2
510    LOCATE SELECT,L
520    PRINT "/"; : SELECT=SELECT-1
530    LOCATE SELECT+1,L
540    PRINT " "; : L=L+1
550  GOTO *LLOOP
560 *EXIT2 : RETURN
```

160行でCLSを使っていますが，その後に数値の3が書いてあります．これは，CLSの機能を決めるパラメータで，どの画面を消すかを指示するものです．プログラム4では，3という値を取っています．これは，テキスト画面とグラフィック画面の両方を消すことを意味します．その他取りうる値は，1と2だけです．

1のときはテキスト画面の中で，CONSOLE文で指定されたスクロール・ウィンドウの中をクリアします．パラメータが1のときのみ，これを省略することができます．また，白黒モードで文字を反転して表示すると，画面は白くなります．次に，パラメータが2のときは，グラフィック画面（但し，白黒の640×200が3ページあるモードでは，アクティブな（現在書き込みの対象となっている）画面のみ）をクリアします．このときは，パラメータの省略はできず，また，カラーモードであれば，バック・グラウンド・カラーで塗りつぶされます．

170行から210行のFOR～NEXTのループで，80個のでたらめな点を打ちます．この点を打つ命令が，PSETです．PSETの書式は，次の通りです．

```
PSET（X，Y），C
```

XとYは，それぞれグラフィック画面の横の座標と縦の座標です．Cはそのドットの色で，パレット番号で指定するのですが，カラーコードと同じものと考えて下さい．したがって，PSETは『座標（X，Y）に，Cで示されるカラー（パレット）番号の点を打つ』という意味です．Cは省略可能で，省略したときにドットの色は，現在，COLOR文で設定されているフォア・グラウンド・カラーになります．

*LOOPでは，星の点滅と，流れ星を発生させるか，させないかを決定しています．*RIGHTと*LEFTでは，右と左に流れる星を作っています．

330行のPRESETの使い方は，PSETと同じく，

```
PRESET（Y，Y），C
```

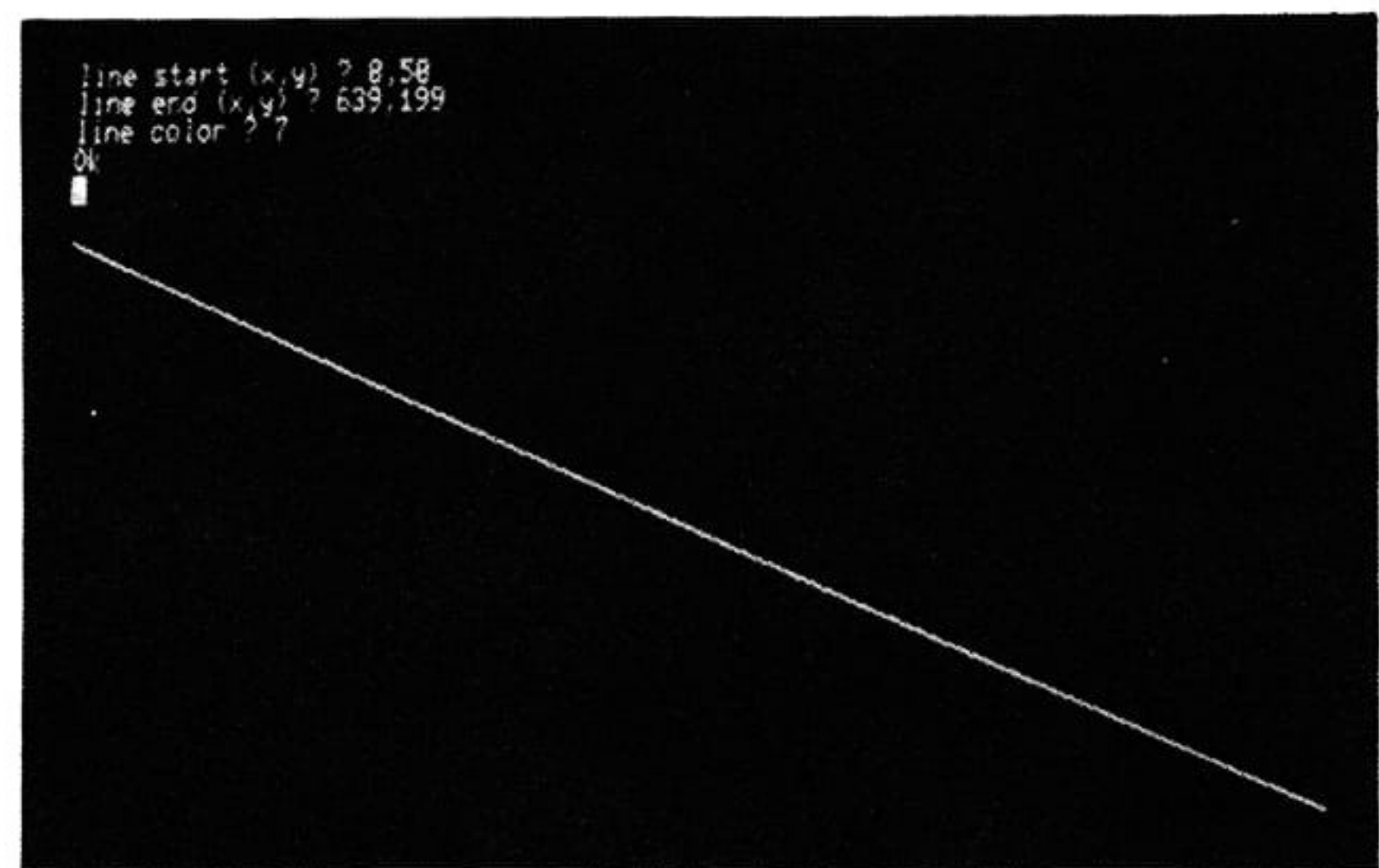
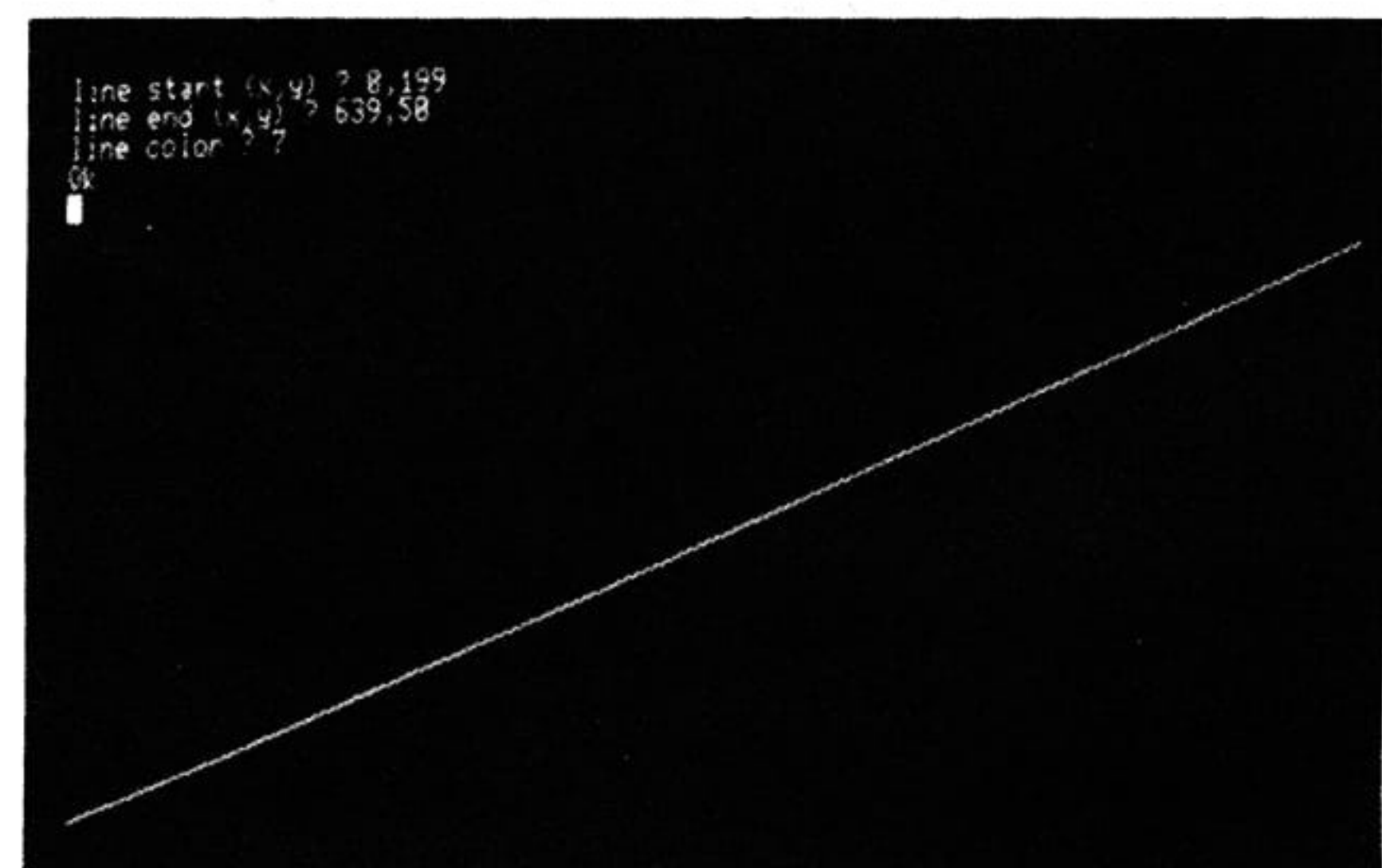
ですが，Cを省略するのが普通です．というのはCを省略した場合，ドットの色がバック・グラウンド・カラーになるからで，そうすれば既に点を打っている所や，何かの図が書いてある所に，PRESETを実行させた場合，そこのドットを消したように（リセットしたように）働きます．このプログラムは，普通の星は，グラフィック画面，流れ星はテキスト画面上に描いていますので，グラフィック画面とテキスト画面が独立していることをもう一度確認して下さい．リストをとってみるなどしてみれば，よく分かるでしょう．

4.3.2 LINE, STEP, POINT—直線を描く—

次に用意されている基本的なグラフィック命令が，線を引くLINEです．4.3.1で言ったように，PSET文を用いれば，LINE文の持っている機能は全て実現できますが，LINE文と比べれば，やはり，そのプログラムは厄介なものとなります．

では，その辺をプログラム5と6で見比べてみましょう．どちらのプログラムも200行までは，線を引く時の引きはじめの座標と引き終わりの座標，さらに，その線の色の入力を促します．200行は，Cに7以下の整数値のみを与えます（理由は，パレット番号が，0から7の整数値しか採らないからです）．プログラム5の方は，PSETを使って線を引いています．変数PXを1ずつ増やしていき，直線を示す式，y＝ax＋bのxにPXを代入して，y（PY）を求めているのが，310行です．その式で求められた座標（PX，PY）に，色Cで点を打っています．

また，240行で行っているのは，310行（＊EQUALX）でDX＝0のとき，すなわち，0で割り算をさせない処理です．これだけややこしいことも，LINE文を用いると，プログラム6のようにシンプルなものとなります．

```
100 '
110 '   Program 5
120 '     line draw demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  LOCATE 0,0,0
170  INPUT "line start (x,y) ";X1,Y1
180  INPUT "line end (x,y) ";X2,Y2
190  INPUT "line color ";C
200  C=C MOD 8
210 '
220 *LINEDRAW
230  IF X1>X2 THEN SWAP X1,X2 : SWAP Y1,Y2
240  IF X1=X2 THEN *EQUALX
250  DX=X2-X1 : DY=Y2-Y1
260  FOR PX=X1 TO X2
270    PY=(PX-X1)*DY/DX+Y1
280    PSET(PX,PY),C
290  NEXT : GOTO *FIN
300 '
310 *EQUALX
320  IF Y1>Y2 THEN SWAP Y1,Y2
330  FOR PY=Y1 TO Y2
340    PSET(X1,PY),C
350  NEXT
360 '
370 *FIN
380 END
```

```
100 '
110 '   Program 6
120 '     line draw demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  LOCATE 0,0,0
170  INPUT "line start (x,y) ";X1,Y1
180  INPUT "line end (x,y) ";X2,Y2
190  INPUT "line color ";C
200  C=C MOD 8
210 '
220 *LINEDRAW
230  LINE(X1,Y1)-(X2,Y2),C
240 END
```

しかし，LINE文は，ただ単に，直線を引くことを容易にするだけではありません．箱を描いたり，さらには，その箱を塗りつぶすといった機能もあります．

プログラム7は，その箱を塗りつぶす機能を用いたものです．ただ，画面を3色に塗り分ける

だけですが，一応フランスの国旗とでも考えて下さい．LINE 文の最後についている『BF』が，箱を塗りつぶすことの指示で，『Box Fill』の意味です．

```
100 '
110 '   Program 7
120 '     Franch flag
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  LINE(0,0)-STEP(213,182),1,BF
170  LINE STEP(1,-182)-STEP(213,182),7,BF
180  LINE STEP(1,-182)-STEP(213,182),2,BF
190  LOCATE 0,23 : PRINT "FRANCE FLAG"
200 END
```

このような四角形の箱を書いたり，それを塗りつぶしたりする際の箱の大きさは，普通の LINE 文で引く線を，その四角形の箱の対角線とした大きさです．そのことは LINE 文で座標の値を変えずに，『何も指示を加えない単なる線』と『Bまたは，BF を付けた箱』を描くことを実行すれば，明らかになります．

さて，プログラム7の LINE 文には，もう一つ見慣れない用法の STEP があります．中には，BASIC の他の命令で見たことがあると，気付いた方もいると思います．それは，FOR～TO～STEP の文での STEP でしょう．LINE 文における STEP は，これとは全く意味が違います．

今までのグラフィック命令では，グラフィック座標で，必ず原点を画面の左上とし，横（X）の座標と縦（Y）の座標を決定すれば，その画面上（座標上）の位置が，決まるものでした．これを『絶対座標による指定』と言います．これに反して，プログラム7に出てくる STEP を用いた座標の指定を，『相対座標による指定』と言います．これは，ある基点からの相対的な移動の量を，横の向きと，縦の向きとで与えるものです．ここで言う『ある基点』というのが，グラフィック座標の SCREEN 文の最後の所に述べた，LP のことです．LP とは，Last referenced Point の略で，『最後にグラフィック命令が，参照した点の座標』という意味です．プログラム7の 160 行を見て下さい．この文の実行に入る前までは，LP がどこであるか，はっきりしません．しかし，LINE 文の最初の座標指定は，絶対座標で指定されていますので，この段階で，既に1つの座標を参照したことになりますから，LP は，（0，0）となります．さて次の座標指定が相対座標を用いています．この直前までは，LP が（0，0）でしたから，この点から横方向に 213，縦方向に 182ドット分移動させた点とを結ぶ線を引きますが，この場合 BF が付いていますから，それを対角線とする四角形の箱を塗りつぶす訳です．結果として得られる図形は，横 214，縦183 の大きさの長方形となります（長方形の大きさが，移動量＋1となっているのは，最初の座標も1つと数えるからです．）

このとき，最後に参照した点は，初めの LP から，横方向 213，縦方向 182ドット移動した所の点ですから，次式のように，

0＋213＝213

0＋182＝182

となって，LP は（213，182）に変わっています．170行は，この LP を基点として，160行と同

様なことを行っています．

このように相対座標による指定ができるのは，LINE 命令だけではありません．既に述べた PSET，PRESETや，これから出てくる CIRCLE，PAINT なども相対座標による指定が可能です．では，相対座標に必要な LP を画面に何の影響も与えず決定することができるでしょうか．画面への影響を与えてよいのなら，

```
PRESET (320, 100)
```

と実行して，LP を (320，100) にしてもよいでしょう．しかし，もし，既にこの点に何かが，描かれていたとすると，この点は PRESET で影響を受けることになります．そこで，次の命令が用意されています．

```
POINT (320, 100)
```

この POINT 文は，『LPを座標（320，100）に移動する』という意味です．

LPは，相対座標による指定だけに参照されるのではなく，LINE 文でプログラム 8 の 360行のような表記――すなわち，線の引きはじめが省略された，絶対座標による指定のとき，LP は，線の引きはじめの座標として用いられます．プログラム 8 は，これを利用して折れ線グラフを描いています．

カラーページ参照

```
100 '
110 '   Program 8
120 '     line demonstration
130 '
140 '
150  SCREEN 0,0
160  WIDTH 80,25 : CONSOLE 0,24,0,1
170  CLS 3
180  LINE(30,5)-(630,185),7,B
190  FOR I=150 TO 510 STEP 120
200    LINE(I,5)-(I,185),7
210  NEXT
220 '
230  FOR J=167 TO 23 STEP -18
240    LINE(30,J)-(630,J),1
250  NEXT
260 '
270  LOCATE 0,0 : PRINT USING "###";100
280  LOCATE 0,12 : PRINT USING "###";50
290  LOCATE 0,23 : PRINT USING "###";0;
300 '
310  FOR C=1 TO 2
320    GOSUB *CHANGE
330    POINT(30,R)
340    FOR RESULT=150 TO 630 STEP 120
350      GOSUB *CHANGE
360      LINE -(RESULT,R),C*2
370    NEXT
380  NEXT
390  A$=INKEY$ : IF A$="" THEN 390
400 END
410 '
420  DATA 62,75,69,83,80,74    :'marks
430  DATA 51,63,65,60,71,72    :'average
440 '
450 *CHANGE
```

```
460  READ R
470  R=185-R*18/10
480 RETURN
```

このプログラムは，ある科目の定期考査の年間のグラフで赤い線が個人，緑の線が平均と考えてください．このグラフの場合，ある個人の一つの科目についてですが，これが複数になったら，折れ線の色だけでなく，その線の形状を変えてみると（破線など）分かり易くなります．

それを実現するためにLINE文に，ライン・スタイルという機能があります．これを設定するためには，16ビットで表すことができる数(&H0～&HFFFF)の値を用います．この値と画面上のドットとの関係は，(図４)のようになっています．つまり，斜線になっている所が表示させるドットであるとすると，そこのビットを１とし，表示させないビットを０としたときの16ビットの値をライン・スタイルとするのです．何故，16進数を用いるかというと，ライン・スタイルの16ビットを４ビットずつ４つに分けたとき，その１つ１つの４ビットは，(表11)を見ると分かると思いますが，16進数に変換し易いからなのです．

カラーページ参照

```
100 '
110 '   Program 9
120 '     line style demonstration
130 '
140 '
150  SCREEN 0,0
160  WIDTH 80,25 : CONSOLE 0,24,0,1
170  CLS 3
180  LINE(30,5)-(630,185),7,B
190  FOR I=150 TO 510 STEP 120
200    LINE(I,5)-(I,185),7
210  NEXT
220 '
230  FOR J=167 TO 23 STEP -18
240    LINE(30,J)-(630,J),1
250  NEXT
260 '
270  LOCATE 0,0 : PRINT USING "###";100
280  LOCATE 0,12 : PRINT USING "###";50
290  LOCATE 0,23 : PRINT USING "###";0;
300 '
310  FOR I=1 TO 2
320    READ LS
330    FOR C=1 TO 3
340      GOSUB *CHANGE
350      POINT(30,R)
360      FOR RESULT=150 TO 630 STEP 120
370        GOSUB *CHANGE
380        LINE -(RESULT,R),C*2,,LS
390      NEXT
400    NEXT
410  NEXT
420  A$=INKEY$ : IF A$="" THEN 420
430 END
440 '
450  DATA &HFFFF
460  DATA 79,59,60,97,43,56
470  DATA 78,57,81,44,95,62
480  DATA 98,46,89,40,76,66
490  DATA &H8888
500  DATA 87,70,92,52,49,53
510  DATA 88,94,51,63,64,72
520  DATA 64,81,67,62,37,47
530 '
540 *CHANGE
550  READ R
560  R=185-R*18/10
570 RETURN
```

プログラム9は，このライン・スタイルを利用して，折れ線グラフを描くものです．表の中で，折れ線の色は，個人の区別をしており，ライン・スタイルは，科目の区別としているものと考えて下さい．この折れ線グラフ以外にも，多くの応用ができるのがLINE文ですから，使い慣れることが大事だと思います．LINE文のまとめとして，その書式を挙げておきます．

LINE（X 1，Y 1）－（X 2，Y 2），パレット番号，B，ライン・スタイル

ドットパターン

1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1　ビットパターン（2進数表記）

&H　D　B　D　B　16進数で表記

斜線部が表示させるドット

ビットパターンを4ビットずつに区切ると，表10より，16進に変換しやすい．覚えてしまえば表11などは不要になります．

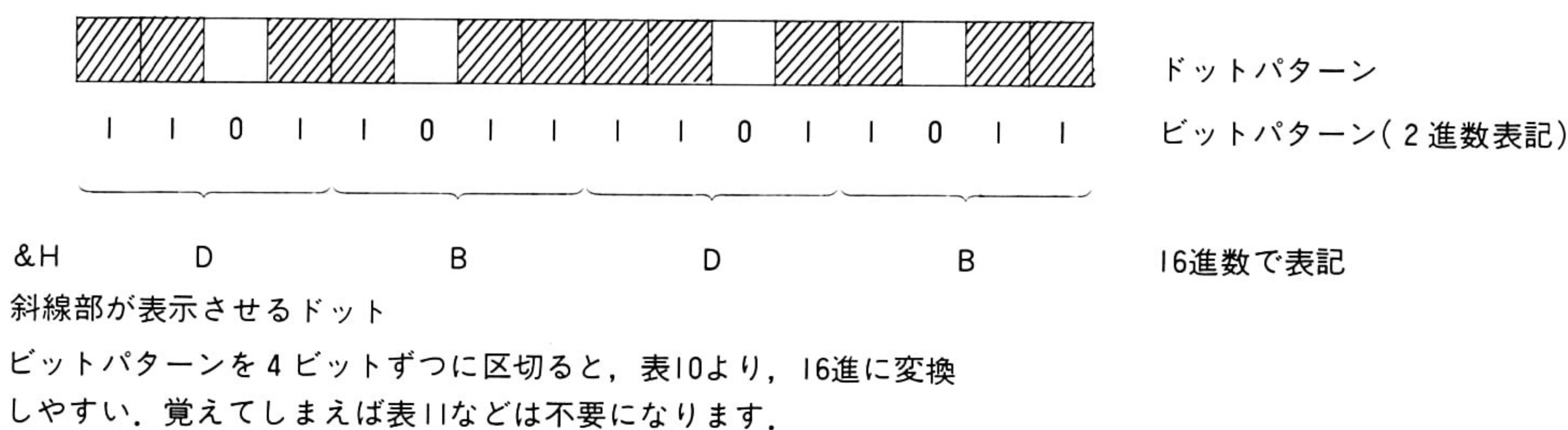



図4　ライン・スタイルと16進数

4ビット 2進数	16進数	4ビット 2進数	16進数
0 0 0 0	0	1 0 0 0	8
0 0 0 1	1	1 0 0 1	9
0 0 1 0	2	1 0 1 0	A
0 0 1 1	3	1 0 1 1	B
0 1 0 0	4	1 1 0 0	C
0 1 0 1	5	1 1 0 1	D
0 1 1 0	6	1 1 1 0	E
0 1 1 1	7	1 1 1 1	F

表11　2進（4ビット）←→16進変換表

4.3.3　CIRCLE—円を描く—

意味の通り，円を描く命令です．円は，その中心となる位置の座標（X，Y）と円の半径Rさえはっきりすれば，描くことができます．これらのパラメータの与え方は，描く色のパレット番号をCとすると，

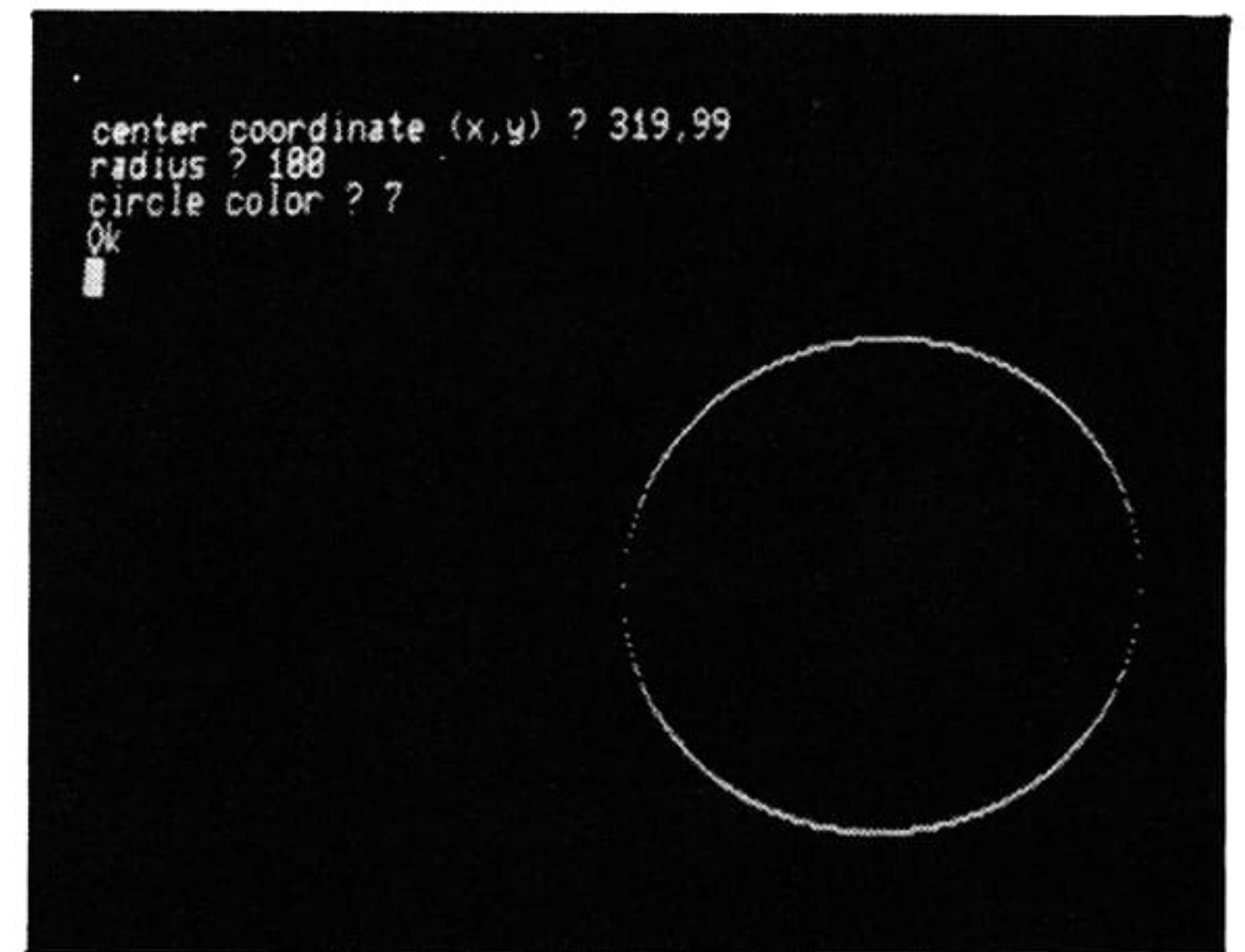
CIRCLE（X，Y），R，C

と書くことができます．

CIRCLE命令の使い方を説明する前に，4.3.2のLINEで行ったように，円をPSET命令を使

って描いてみます．読者の中には，『CIRCLE 命令があるのだから，わざわざ PSET で……』とお考えの方もいるかと思いますが，実際，円というものがどのような方法で描かれていくのか，ということを知っておくのは，今後さまざまな図形を描こうとする時の助けとなり，有意義なことだと思います．

円を描く方法（もちろん CIRCLE 命令を用いない方法）は，いくつかありますが，その思いついたものだけ，プログラム 10，11，12に挙げました．

```
100 '
110 '  Program 10
120 '    circle demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  INPUT "center coordinate (x,y) ";CX,CY
170  INPUT "radius ";R
180  INPUT "circle color ";C
190 '
200  FOR X=CX-R TO CX+R
210    Y=CY*2+SQR(R*R-(X-CX)*(X-CX))
220    PSET(X,Y/2),C
230    Y=CY*2-SQR(R*R-(X-CX)*(X-CX))
240    PSET(X,Y/2),C
250  NEXT
260 END
```

プログラム10は，円の中心の座標を（a，b）としたときの，円を表す式，

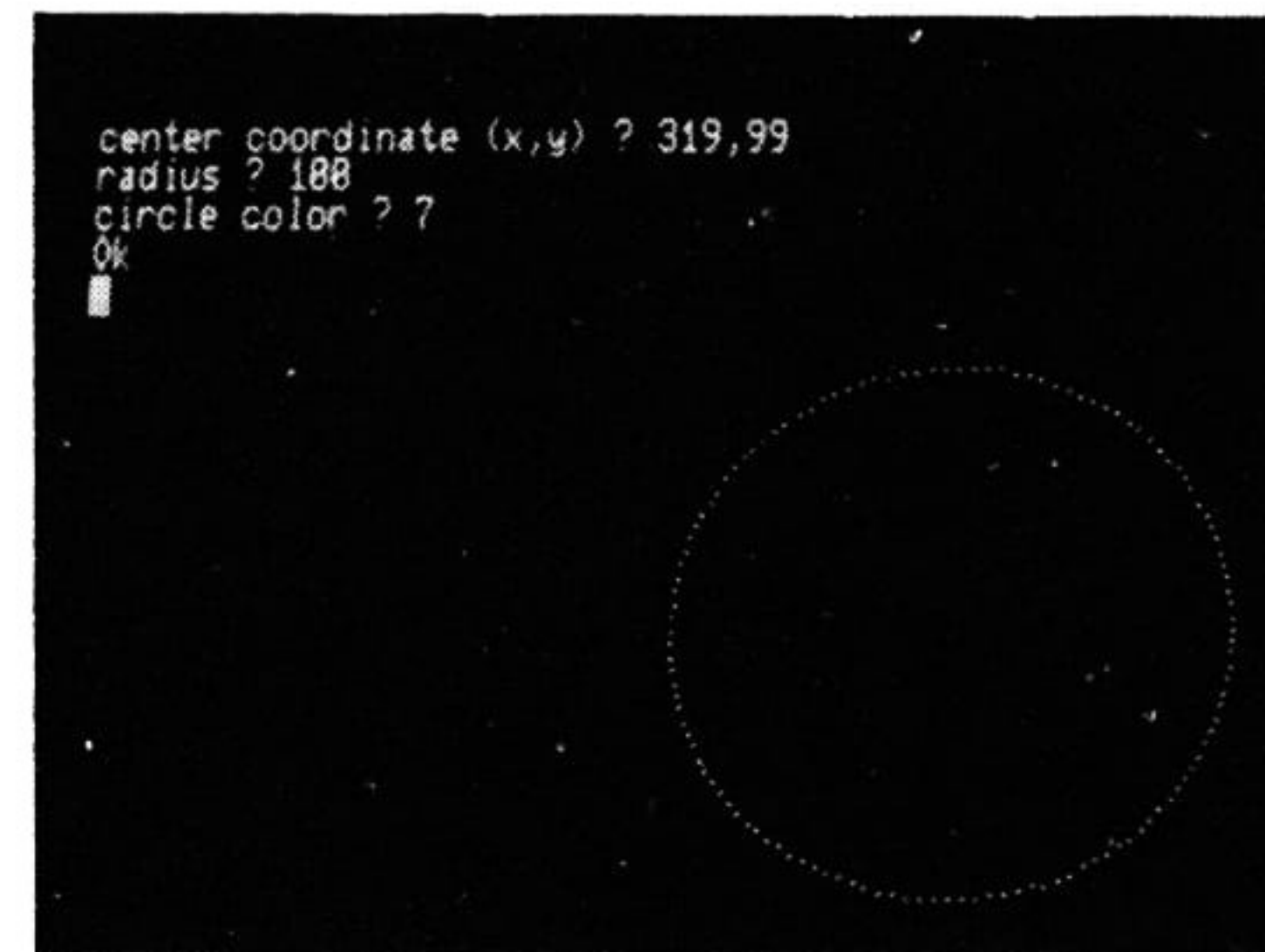
$$(x-a)^2+(y-b)^2=r^2 \qquad r\text{は半径}$$

を用いています．この式の中で変化するのは，（変数は）x と y の 2 つです．そこで式を，次のように変形させます．

$$(y-b)^2=r^2-(x-a)^2$$
$$y-b=\sqrt{r^2-(x-a)^2}$$
$$y=\sqrt{r^2-(x-a)^2}+b$$

この式で，x の値を変化させれば，それに対応した y の値が得られます．実際，プログラム10の中では，x の値が中心の x の座標 CX より半径Rだけ小さい値 CX－R から，同じく半径Rだけ大きい値CX＋Rまで，1ずつ増加されて変化していきます（200行以降）．210行と230行では，中心を通って横軸に平行な直線を軸として，対称な位置を求めています．なお，グラフィックのドットの並び方が縦と横で1対1の間隔になっていないので，画面上できれいな円を描くために，幾つかの補正（210行と230行で，CY を 2 倍していることと，220行と240行でYを 2 で割っていること）を行っています．

```
100 '
110 '  Program 11
120 '    circle demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  INPUT "center coordinate (x,y) ";CX,CY
```

```
170  INPUT "radius ";R
180  INPUT "circle color ";C
190 '
200  FOR P=0 TO 2*3.1415 STEP .05
210    X=R*COS(P)+CX
220    Y=R*SIN(P)+CY*2
230    PSET(X,Y/2),C
240  NEXT
250 END
```

プログラム11は，もう少し高級に三角関数を用いています．(図６)を見て下さい．図のように記号を付けたとき，

sin（θ)＝c／a，cos（θ)＝b／a

となる関数が，サインとコサインです．このときに，θは，ラジアンという角度の単位を用います．ラジアンというのは，（図８）に示されているように，180度をπ（＝3.14159……）に換算した値で，弧度といいます．この弧度は，次式で求められます．

（弧度)＝（度）＊π／180

但し，π＝3 .14159265358979……

このようなラジアンによる角度の取り扱いを弧度法と言います．弧度法の説明は，専門書に任せることにして，今は少なくとも（図８）と上式の関係を知っておいて下さい．さて，サインとコサインのことに話を戻しますが，先程の式（図６）については，

C＝a＊sin（θ)，b＝a＊cos（θ)

と書き直すことができます．したがって，角度（ラジアン）と斜辺の長さaが，既知ならばbとcの長さも求められます．このことを利用すれば，半径Rが一定のとき角度Pを０から２π(一周分）まで0.05きざみで変化させて，その度にコサインとサインを用いて，x方向とy方向の座標を求めて点を打っていけば，円を描くことができます．210行と220行でCXとCY＊２を加えているのは，中心の位置を所定の位置にずらす計算です．

```
100 '
110 '   Program 12
120 '     circle demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  INPUT "center coordinate (x,y) ";CX,CY
170  INPUT "radius ";R
180  INPUT "circle color ";C
190 '
200  X=R : Y=0
210  FOR P=0 TO 400
220    PX=X+CX : PY=(Y+CY*2)/2
230    PSET(PX,PY),C
240    X=X-Y/64 : Y=X/64+Y
250  NEXT
260 END
```

図6　プログラム11の円の描き方(a)

π=3.14159
弧度法のときの角度の値は，
(弧度)=(度)＊π/180で求められる

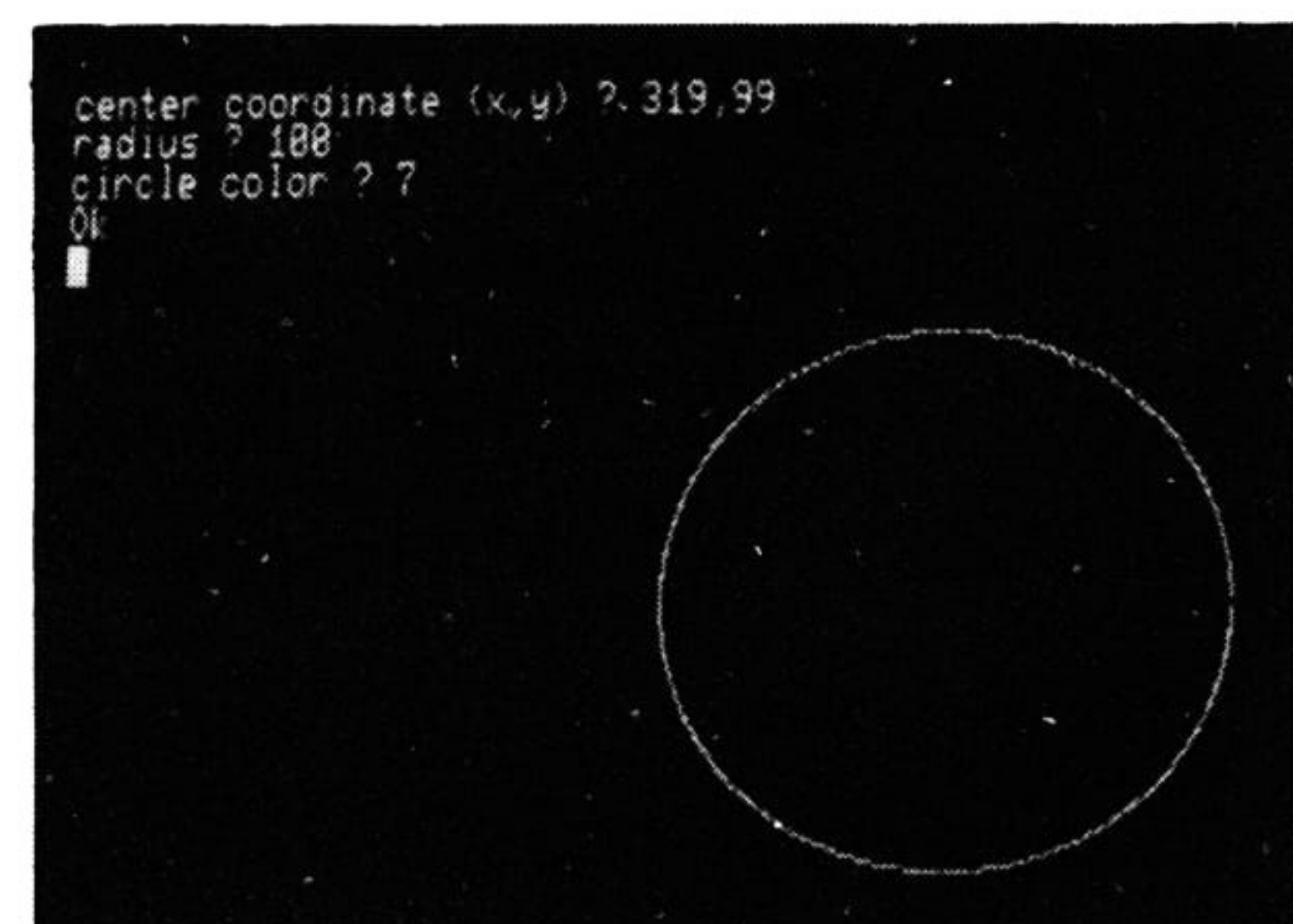
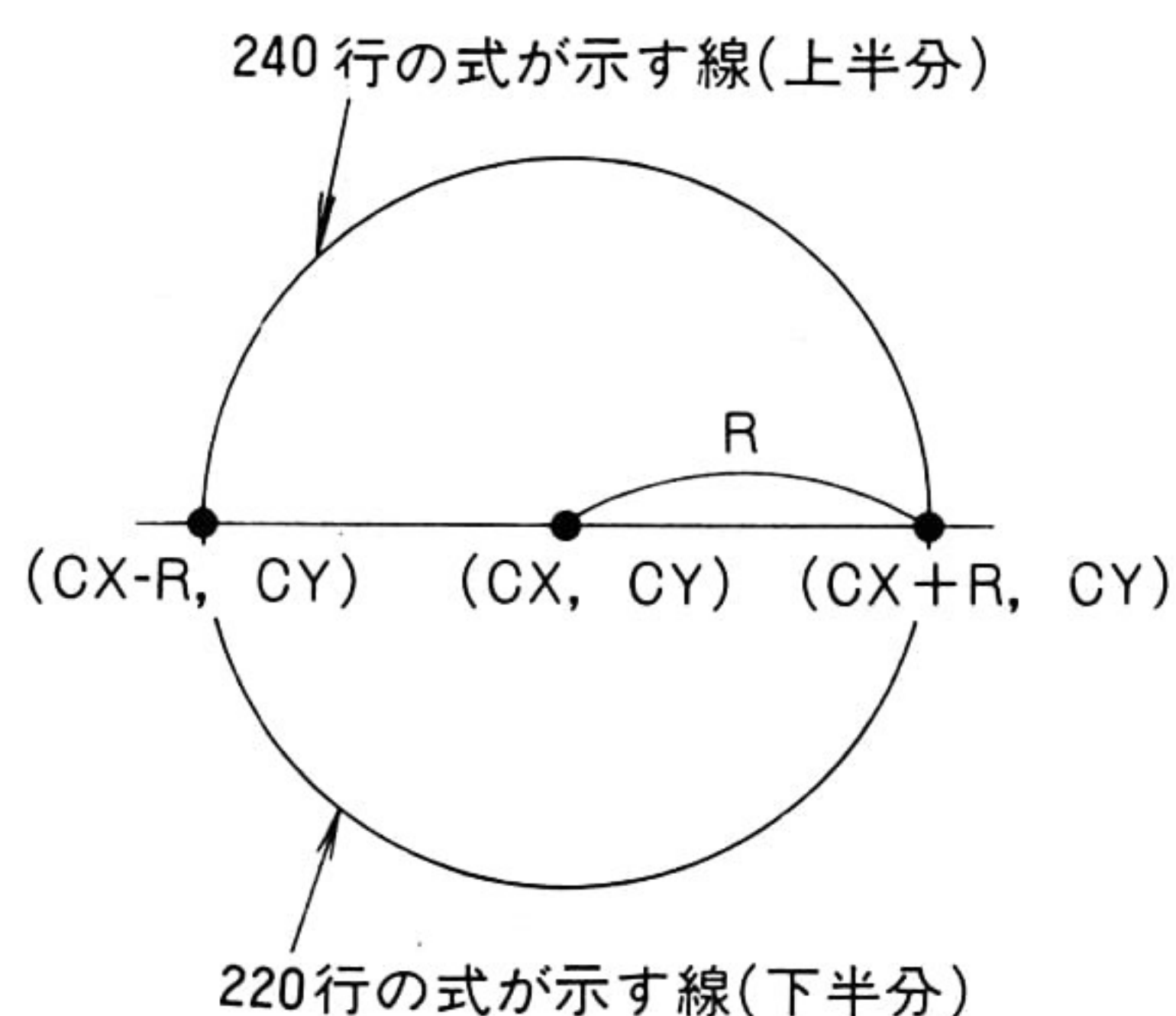



図5　プログラム10の円の描き方

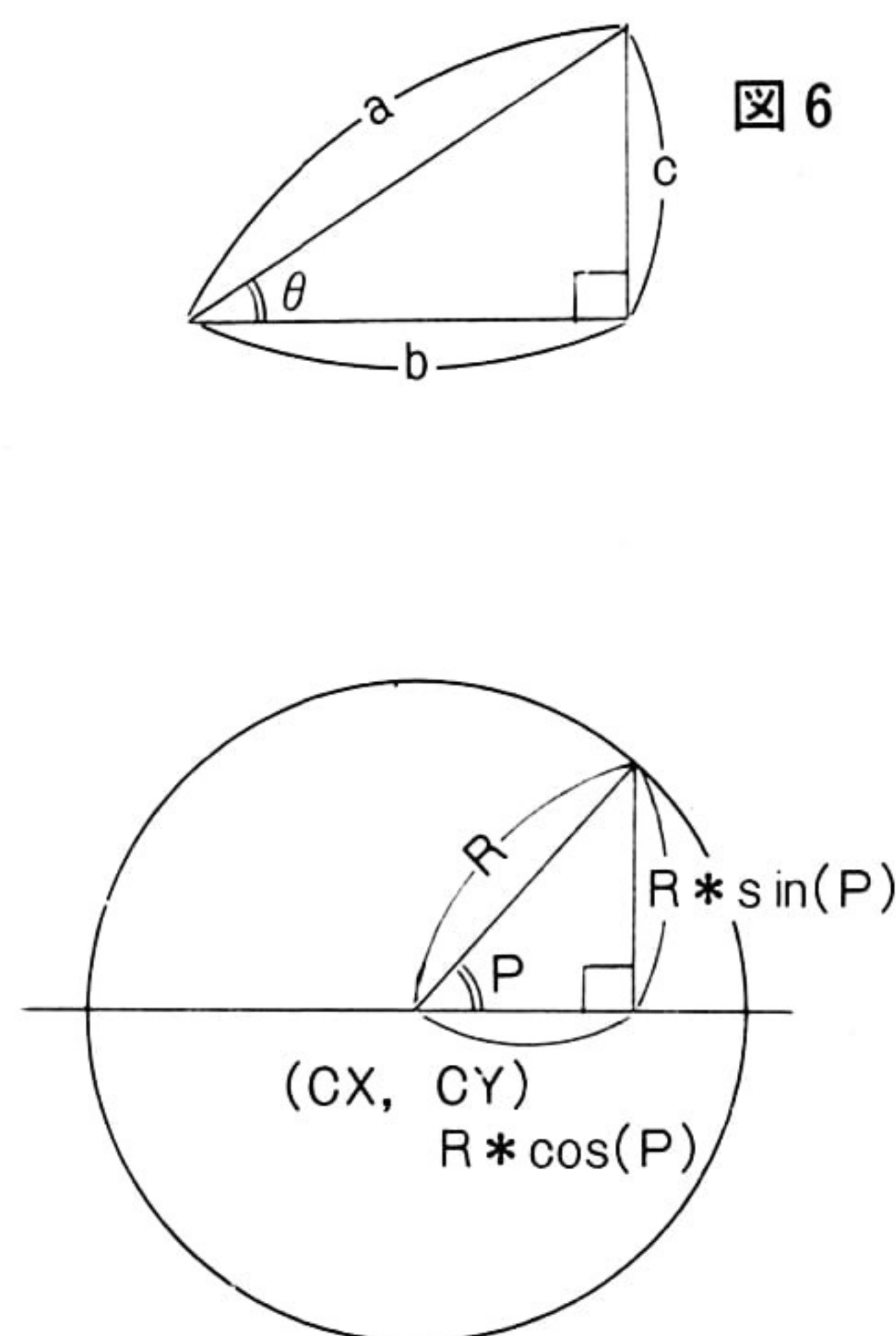



図7　プログラム11の円の描き方(b)

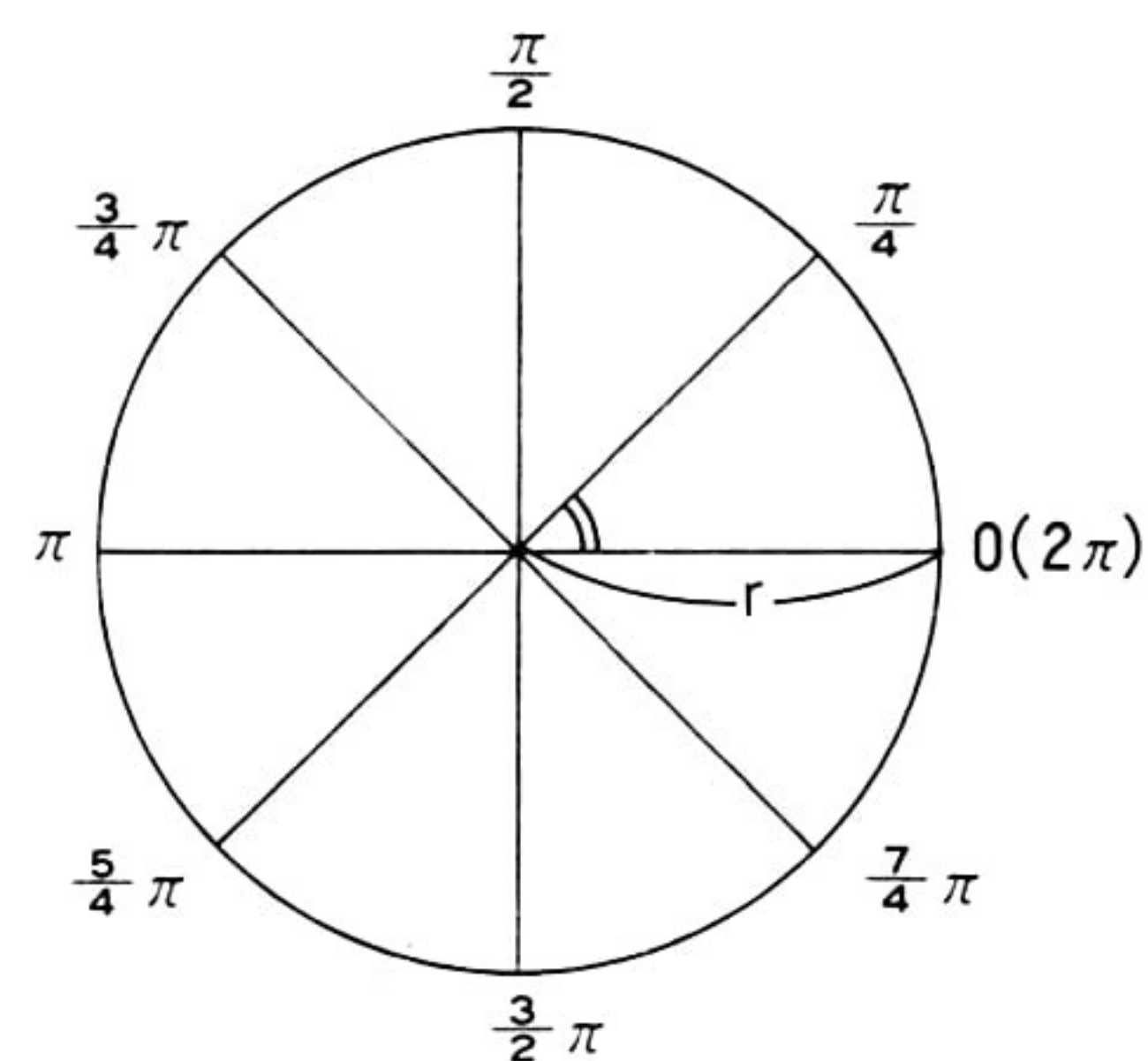



図8　弧度法における角度の値

プログラム12は，行列という考え方が入っています．この行列という考え方は，うまく利用すると図形の処理で，大きな力を発揮します．(ここでは，関係ありませんが，3次元を2次元に変換するのにも，行列は用いられるのです)．プログラム12で用いている行列は，いわば平面(2次元)図形を他の平面図形へ移しているといってよいでしょう．すなわち，点という平面図形を(CX，CY)を中心に回転移動している訳です．プログラム12では，割り算を用いていますが，実際，原点を中心に回転移動させる行列は，

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

で与えられます．例えば，(図9)のように，座標(x，y)を，原点を中心に角度θ(θの値は，正とします)だけ，左回りに回転させた座標(x，y)に移すことを意味します．もちろん，θを負の値とすれば，右回りに回転した位置に移ります．

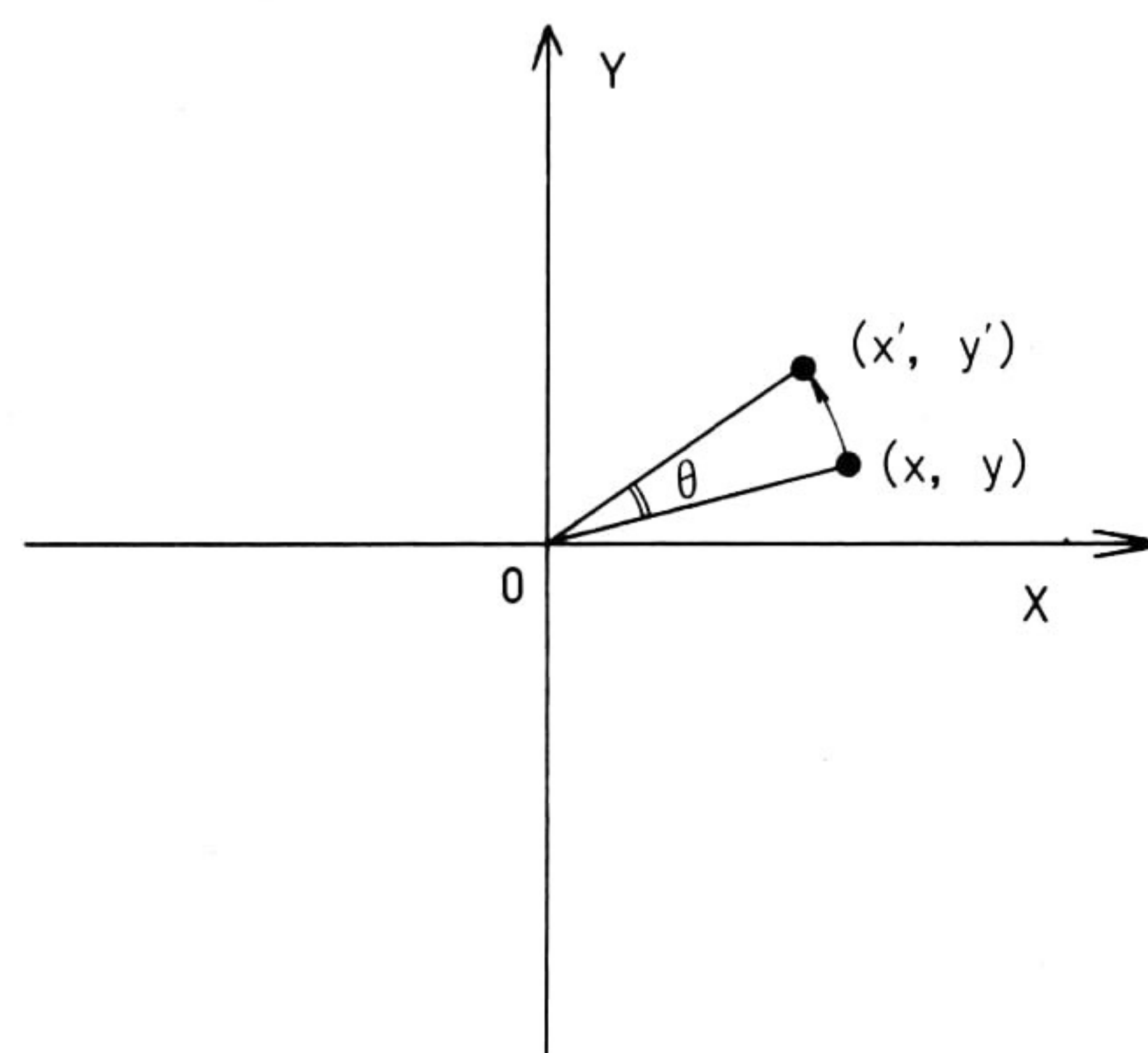
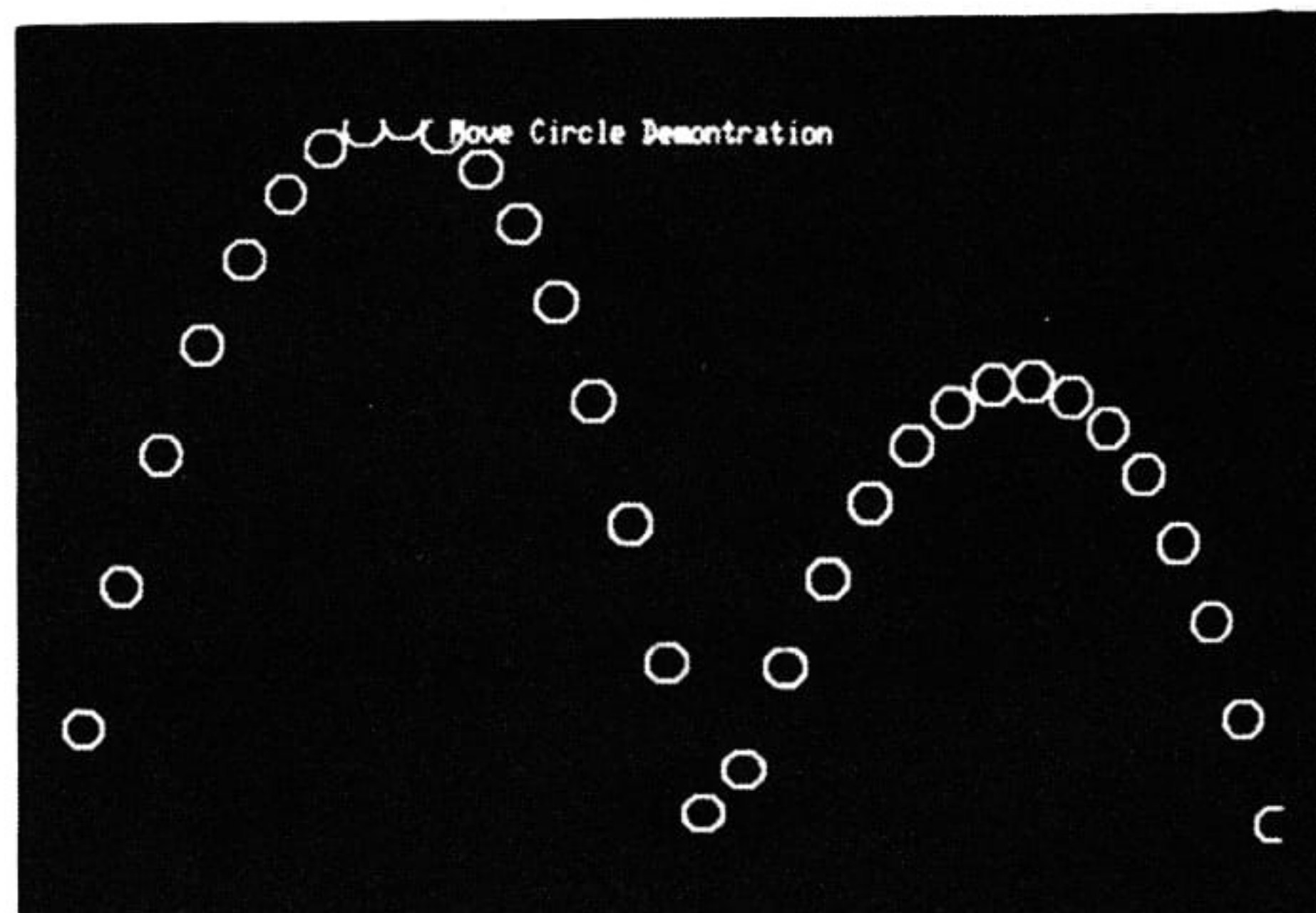



図9　座標の回転

しかし，θ（回転する角度）を0に近い定数とすれば，cos θは，ほぼ1になり sin θは1以下の定数となるので，割り算だけで消むのです．（但し，グラフィック画面のＹ軸は，下向きが正なので，プログラム12を実行させると右回りに描いていきます．）

これらと比較すると，速度の面だけをとってもやはり CIRCLE 文は，重宝です．CIRCLE 文で小さい円を描いて，これを動かすプログラム，すなわち，単純なアニメーションを行ってみましょう．プログラム 13で用いているアニメーションの手法は，白黒 640×200 のモードで，3ページのうちのプレーン1とプレーン2に対して，交互に表示，書き込み，消去を行っている訳です．その『交互』という作業を実現しているのが，SCREEN 文です．表示させている画面と書き込む画面を別にしているため，円を描いている途中の過程は見なくても済みます．また，画面に1回書いた円を消すのに，CLS 命令では時間がかかり過ぎるので，前と同じ所に黒で円を描いて消しています．＊FUNCTION 以降の文を書き換えることによって，その軌跡を変えることができます．

```
100 '
110 '   Program 13
120 '     animation (page demonstration)
130 '
140 '
150  CLS:LOCATE 25,0
160  PRINT "Move Circle Demontration"
170  SCREEN 0,1
180  CLS 2
190  SCREEN 1,0,1
200  A=0
210  FOR X=5 TO 640 STEP 10
220    B=A:A=(A+1) MOD 2
230    SCREEN ,,B,A+1
240    BY=Y
250    GOSUB *FUNCTION
260    CIRCLE(X,Y),10
270    SCREEN ,,A,B+1
280    CIRCLE(X-10,BY),10,0
290  NEXT
300 END
310 '
320 *FUNCTION
330  IF X<340 THEN Y=(X-170)*(X-170)/145
              ELSE Y=(X-490)*(X-490)/174+70
340 RETURN
```

CIRCLE 命令は，ただ円を描くだけでなく，描く範囲を指定すれば円弧や扇形を描くこともできます．範囲の指定は，開始角度と終了角度をラジアン（角度の単位）で指定してやります．この角度の値は，両者とも－2πから＋2πの範囲内である必要があります．この角度指定が正であった場合，（0から2πであったとき）指定範囲内に円弧を描きます．負であった場合，指定範囲内に（絶対値をとって，正の値にしたとき得られる範囲内）に扇形を描きます．書式は次のようになります．

CIRCLE（X，Y），R，C，開始角度，終了角度

例えば，

CIRCLE (320, 100), 80, 7, ST, EN

の命令を考えたとき，この文の前に，

ST＝3.1415＊7／4
EN＝3.1415／4

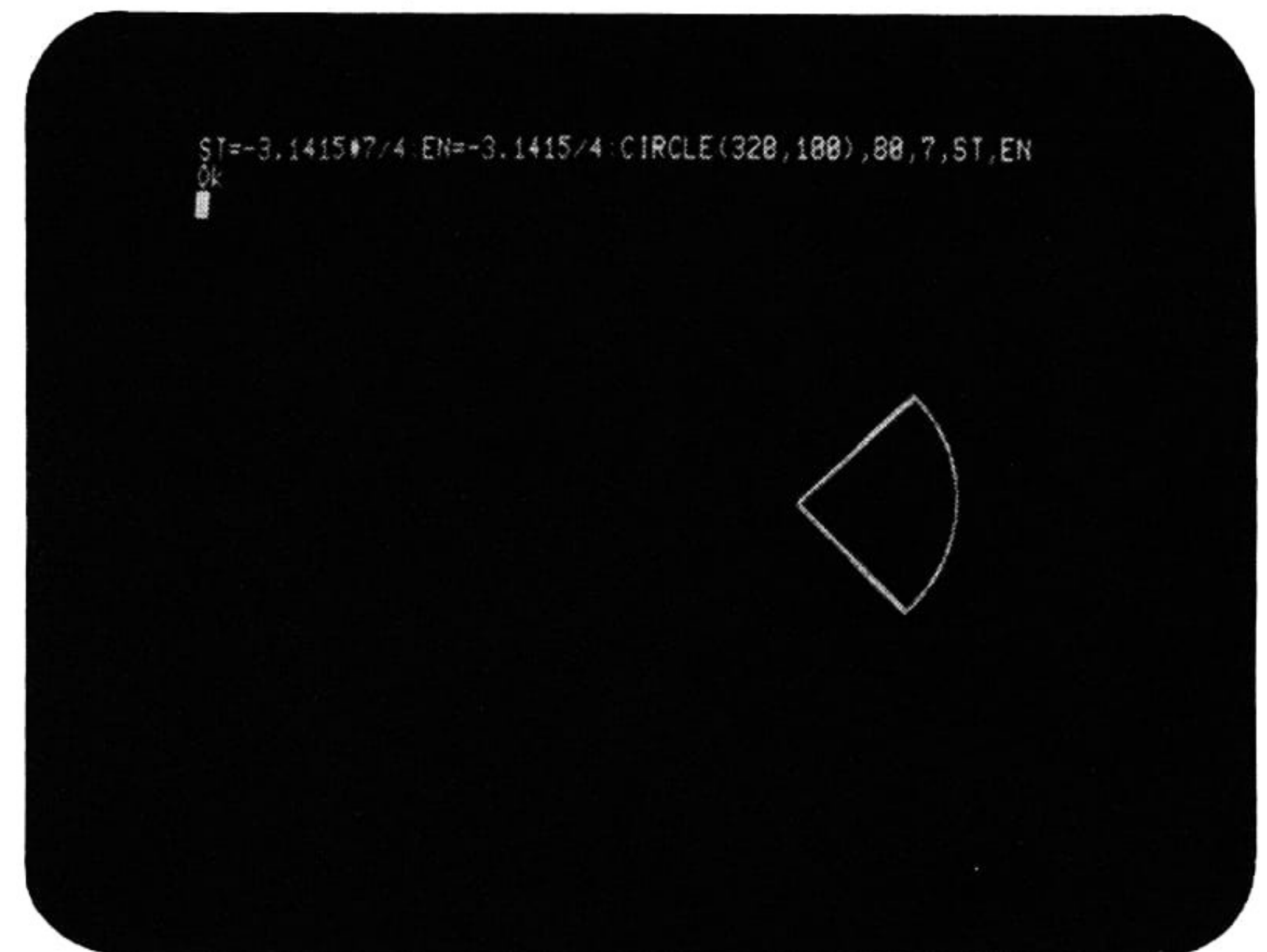
を実行すると，（写真１）の様に，円弧を描きます．

ST＝3.1415＊7／4
EN＝－3.1415／4

をCIRCLE命令の前に実行した時は，（写真２）の様に扇形を描きます．図からも分かるように，同じ範囲に描いています．プログラム14は，上に述べたことのデモンストレーションで，ただ模様を描くプログラムです．

写真 1

写真 2

カラーページ参照

```
100 '
110 '   Program 14
120 '     circle demonstration
130 '
140 '
150  SCREEN 0,1 : CLS 3
160 '
170  ST=3.1415*7/4 : EN=3.1415/4
180  FOR RY=20 TO 150 STEP 60
190    FOR RX=600 TO 0 STEP -30
200      CIRCLE(RX,RY),40,1,-ST,-EN
210      CIRCLE STEP(-10,0),40,2,ST,EN
220      CIRCLE STEP(-10,0),40,4,ST,EN
230    NEXT
240  NEXT
250 '
260  ST=3.1415*3/4 : EN=3.1415*5/4
270  FOR RY=50 TO 170 STEP 60
280    FOR RX=25 TO 680 STEP 30
290      CIRCLE(RX,RY),40,1,ST,EN
300      CIRCLE STEP(10,0),40,4,ST,EN
310      CIRCLE STEP(10,0),40,2,-ST,-EN
320    NEXT
```

```
330  NEXT
340 END
```

さらにもう一つ，CIRCLE 文にはパラメータを与えることができます．これは比率と言い，その値は（垂直方向の半径）／（水平方向の半径）で与えられます．すなわち，この値を設定することによって，だ円を描くことも可能であるということです．パラメータの指定は，終了角度の次にカンマで区切って行います．省略された場合には，円が描かれるので，1.0をその値とするはずなのですが，640×200 のモードでは，縦・横比が1でないため，それを補正するために，0.5となっています．640×400のモードでは，1.0となっています．プログラム15は，この比率をコサインによって与えています．このプログラムを作っていて分かったのですが，比率に負の値を与えても，エラーにはなりません．かといって，絶対値をとっている訳でもないので，思わぬ楕円を描きます．試してみたい方は，190行のABSを取って実行してみると違いが分かるでしょう．

CIRCLE 文を実行した後，LP は，CIRCLE 文で指定した中心に移ります．もちろん，中心の座標の指定は，STEPを付けて相対座標指定でも可能です．

```
100 '
110 '   Program 15
120 '     circle demonstration
130 '
140 '
150  CONSOLE ,,0
160  SCREEN 0,0 : CLS 3
170  X=10 : C=1
180  FOR I=0 TO 3.142 STEP .05
190    P=ABS(COS(I))
200    CIRCLE(X,100),50,C,,,P
210    X=X+10
220    C=(C MOD 7)+1
230  NEXT
240 END
```

カラーページ参照

4.3.4 PAINT—面を塗る—

さて次の命令は，閉じた平面を塗りつぶす命令です．この命令の書式は次の通りです．

PAINT（X，Y），領域色，境界色

座標（X，Y）は，塗り始める所を示します．これは，閉じた平面内になければなりません．画面のふちは閉じた平面と見なされるので，この場合線でわざわざ囲むことはありません．ここに挙げた書式は，絶対座標指定ですが，もちろん STEP を添えれば，相対座標指定になります．

領域色は塗る色のことで，パレット番号で指定します．今は，カラーコードで指定していると考えてもらえれば結構です．これを省略すると，COLOR 文で設定しておいた，フォア・グラウンド・カラーが用いられます．

境界色はどんな色で囲まれた部分を塗るのかを，パレット番号で指定します．これを省略した場合は，領域色と同じパレット番号が用いられます．

では，この PAINT 文を用いて，プログラムを組んでみましょう．プログラム 16は，（表４）

及び（図2）の関係を実際視覚的に摑んでもらおうという訳です．（図2）と見比べてみて下さい．

カラーページ参照

```
100 '
110 '   Program 16
120 '     paint demonstration
130 '
140 '
150  CONSOLE ,,0
160  SCREEN 0,0 : CLS 3
170  FOR I=1 TO 3
180    READ CX,CY
190    CIRCLE(CX,CY),120,7
200  NEXT
210 '
220  FOR I=1 TO 7
230    READ PX,PY
240    PAINT(PX,PY),I,7
250  NEXT
260 END
270 '
280  DATA 320,60,234,130,406,130
290  DATA 440,74,200,74,320,121
300  DATA 320,1,406,71,234,71,320,90
```

プログラム17は4.3.3の CIRCLE 文を用いて，扇形を書き，それを塗りつぶして円グラフを描きます．

カラーページ参照

```
100 '
110 '   Program 17
120 '     円 ｸﾞﾗﾌ & ｵﾋﾞｸﾞﾗﾌ demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  LOCATE 40,0 : PRINT "円 ｸﾞﾗﾌ & ｵﾋﾞｸﾞﾗﾌ demonstration"
170  ST=.00001 : CY=5
180  CIRCLE(200,60),102,7
190  RESTORE *SHARE
200  FOR C=1 TO 5
210    READ D : EN=ST+D/100*3.14*2
220    CIRCLE(200,60),100,C,-ST,-EN
230    PX=50*COS(D/100*3.14-EN)+200
240    PY=25*SIN(D/100*3.14-EN)+60
250    PAINT(PX,PY),C,C
260    ST=EN
270    LOCATE 46,CY : PRINT "■ : ";D;" %"
280    COLOR@(46,CY)-(47,CY),C
290    CY=CY+1
300  NEXT
310 '
320  LINE(120,180)-(520,180),7
330  FOR I=120 TO 520 STEP 40
340    LINE(I,175)-(I,180),7
350  NEXT
360  LOCATE 0,23
370  PRINT SPC(14);"0%";SPC(23);"50%";SPC(21);"100%"
380 '
390  LINE(119,139)-(521,171),7,B
400  RESTORE *SHARE
410  POINT(120,170)
420  FOR C=1 TO 5
430    READ D
440    LINE STEP(0,-30)-STEP(D*4,30),C,BF
450  NEXT
460  IF INKEY$="" THEN 460
470 END
480 '
490 *SHARE
500  DATA 25,5,40,13,17
```

ついでにLINE文のBFのオプションを用いて，帯グラフも描いてみました．ある会社の総売り上げにおける，5つの製品それぞれの占める割合，とでも考えて下さい．このようにPAINT文は，グラフ作成にもその力を遺憾なく発揮します．

ここで，次の疑問が浮かんでくる人があるかもしれません．「PAINT文では，カラーコードにある色しか表現できないのでしょうか？」実際，画用紙などに絵を描くとき，たった8色を混ぜもせず使うということは，絵を描いた経験のある人(経験がないという人はいないと思いますが)なら，誰しも考えられないことでしょう．もちろん，こんな話を持ち出すのですから，色を混ぜることをPAINT文で実現できるのは，予想がつくでしょう．これを実現してくれるのがタイリングであり，その時の書式は次のようなものになります．

PAINT（X，Y），タイル・ストリング，境界色，バック・グラウンド・ストリング

（X，Y）は，タイリングを始める座標で，STEPを付ければ，相対座標指定になります．次にくるパラメータのタイル・ストリングは，初めてお目にかかる言葉です．これがタイリングによって，模様の付いたタイルや，中間色である平面を埋めるときの要素となります．ストリングという言葉に表されるように，このタイル・ストリングは，ストリング変数を用います．したがって，要素は文字列と同じ処理ができます．逆に言えば，このタイル・ストリングは，文字列と同じ処理をしなければいけないのです．

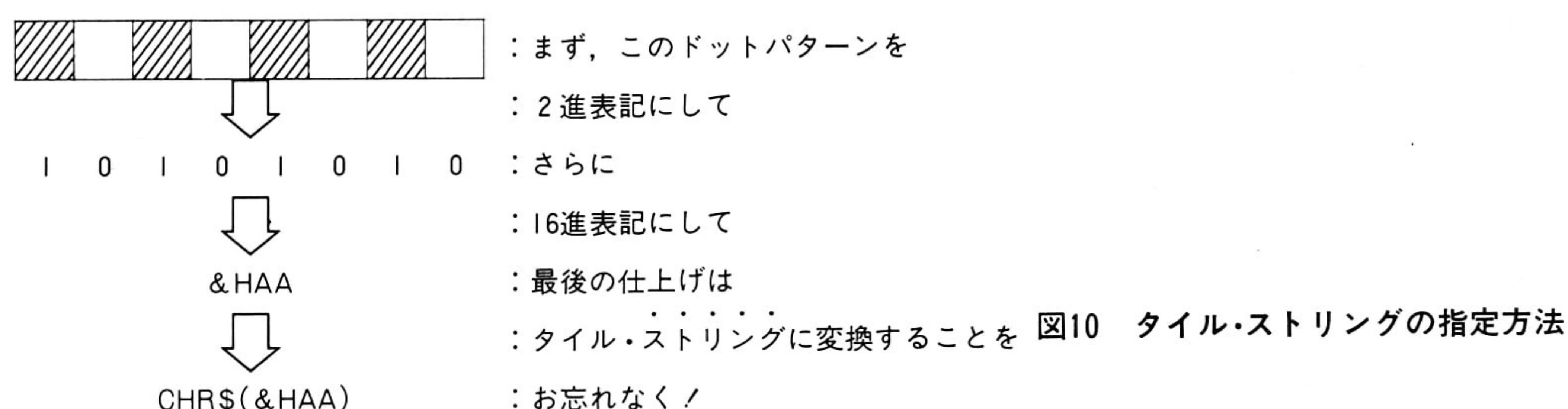
タイルの大きさは，横方向は8ドット分と決まっていますが，縦方向は，タイル・ストリングの長さで変えることができます．縦方向の長さを，nドットとりたいとき，

白黒モードで　　n 文字
カラーモードで　3・n文字

を用意する必要があります．この模様の決め方は，白黒モードとカラーモードでは異なります．

白黒モードは，横8ドットを1文字として扱います．この8ドットの表現の仕方は，LINE文のライン・スタイルと同じ考え方でいいのです．但し，ライン・スタイルは，16ビット単位で指定しましたが，タイル・ストリングは，8ビット単位で指定します．(図10)にその変換を順序立てて示します．(図10)で大切なのは，最後に文字の形にすることです．この様な文字を文字列のたし算を用いて，幾つかつなげれば，白黒モードのときは模様に見えるか，灰色っぽく見えるでしょう．例をプログラム18に挙げておきます．

```
[▨][ ][▨][ ][▨][ ][▨][ ]   ：まず，このドットパターンを
        ⇩                  ：2進表記にして
 1 0 1 0 1 0 1 0           ：さらに
        ⇩                  ：16進表記にして
      &HAA                 ：最後の仕上げは
        ⇩                  ：タイル・ストリングに変換することを
   CHR$(&HAA)              ：お忘れなく！
```

図10　タイル・ストリングの指定方法

```
100 '
110 '   Program 18
120 '     Tiling demonstration
130 '
140 '
150  CONSOLE ,,0
160  SCREEN 1,0,0,1 : CLS 3
170  J=39
180 '
190  FOR I=127 TO 639 STEP 128
200    LINE(0,0)-(I,J),7,B
210    GOSUB *TILING
220    PAINT(I-10,J-10),TILE$,7
230    J=J+40
240  NEXT
250 END
260 '
270 *TILING
280  TILE$=""
290  READ P,Q
300  ON P GOTO 320,330,340,350
310  A$=CHR$(&H55) : B$=CHR$(&HAA) : GOTO *EXIT
320  A$=CHR$(&H33) : B$=CHR$(&HCC) : GOTO *EXIT
330  A$=CHR$(&HF) : B$=CHR$(&HF0) : GOTO *EXIT
340  A$=CHR$(&HBB) : B$=CHR$(&HEE) : GOTO *EXIT
350  A$=CHR$(&H11) : B$=CHR$(&H44)
360  *EXIT
370   FOR L=1 TO Q
380     TILE$=TILE$+A$+B$
390   NEXT
400 RETURN
410 '
420  DATA 3,1,0,1,1,2,2,4,4,1
```

カラーページ参照

カラーモードの場合は，少々白黒モードのときと違った考え方をします．まず違うことは，3文字で横8ドットを表すことです．この3文字は画面の所で説明した3つのプレーンに，それぞれ1文字ずつ対応しています．したがって，この3文字は最初の文字から順に，青，赤，緑のドットパターンを示していることになります．例えば，

CHR$ (&HAA)+CHR$ (&H55)+CHR$ (0)

とすると，これは，(図11) のようなパターンを8ドット上に合成し表現します．画面に出すと，青と赤が交互に出る縞模様になりますが，その縞は，細かいため紫色のように見えます．(図12) のようにすれば，緑と白を混ぜたような効果が得られます．このタイリングを利用して隣接する横方向の2つのドットを，論理的な1ドットと考えることで，27の色を表現することができます．たとえば，〝赤と白のドットを組み合わせてピンク〟といった具合です．ちなみに論理的な1ドットに含まれる実際のドット数 (Bn) と表現可能な色の数 (Cn) の関係は，次のようになります．

$Cn=(Bn+1)^3$　　ただし3はカラーバンクの数 (BRG)

Blue →CHR$(&HAA)
+
Red →CHR$(&H55)
+
Green→CHR$(0)

図11 カラーモードにおけるタイル・ストリングの考え方①

Blue →CHR$(&H55)

＋

Red →CHR$(&H55)

＋

Green→CHR$(&HFF)

‖

緑 白 緑 白 緑 白 緑 白

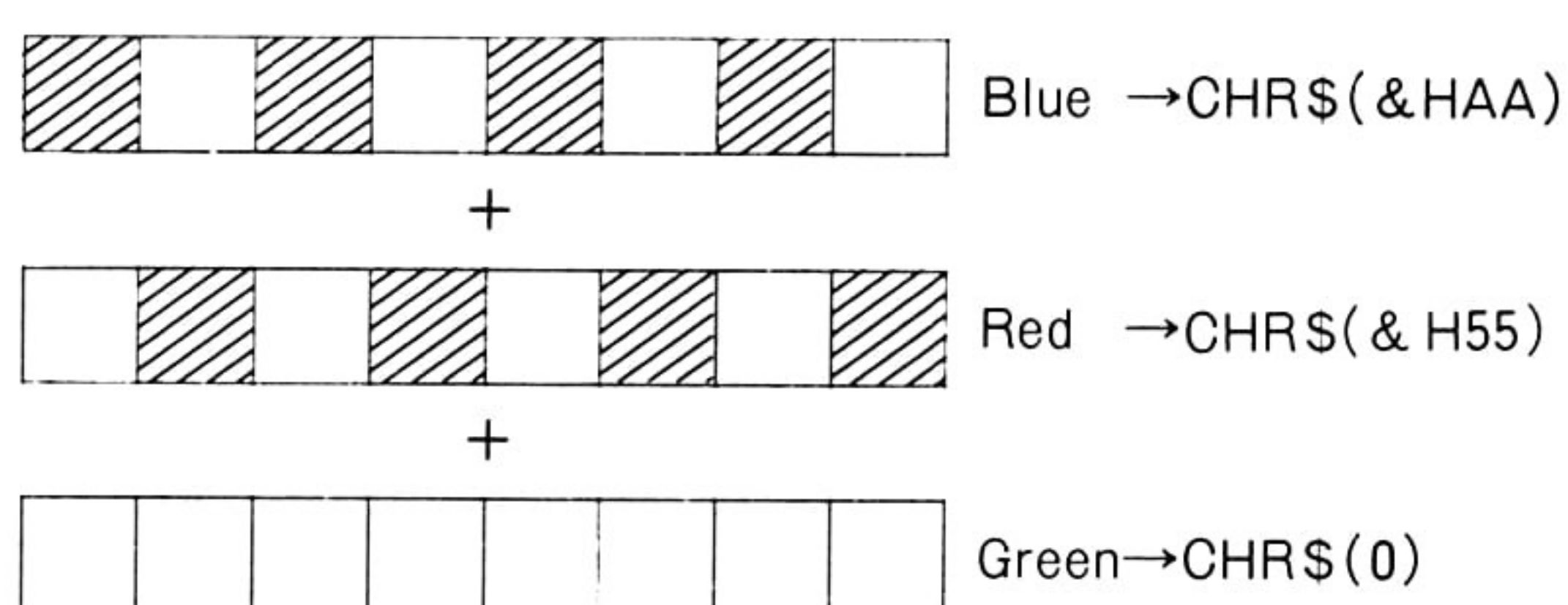
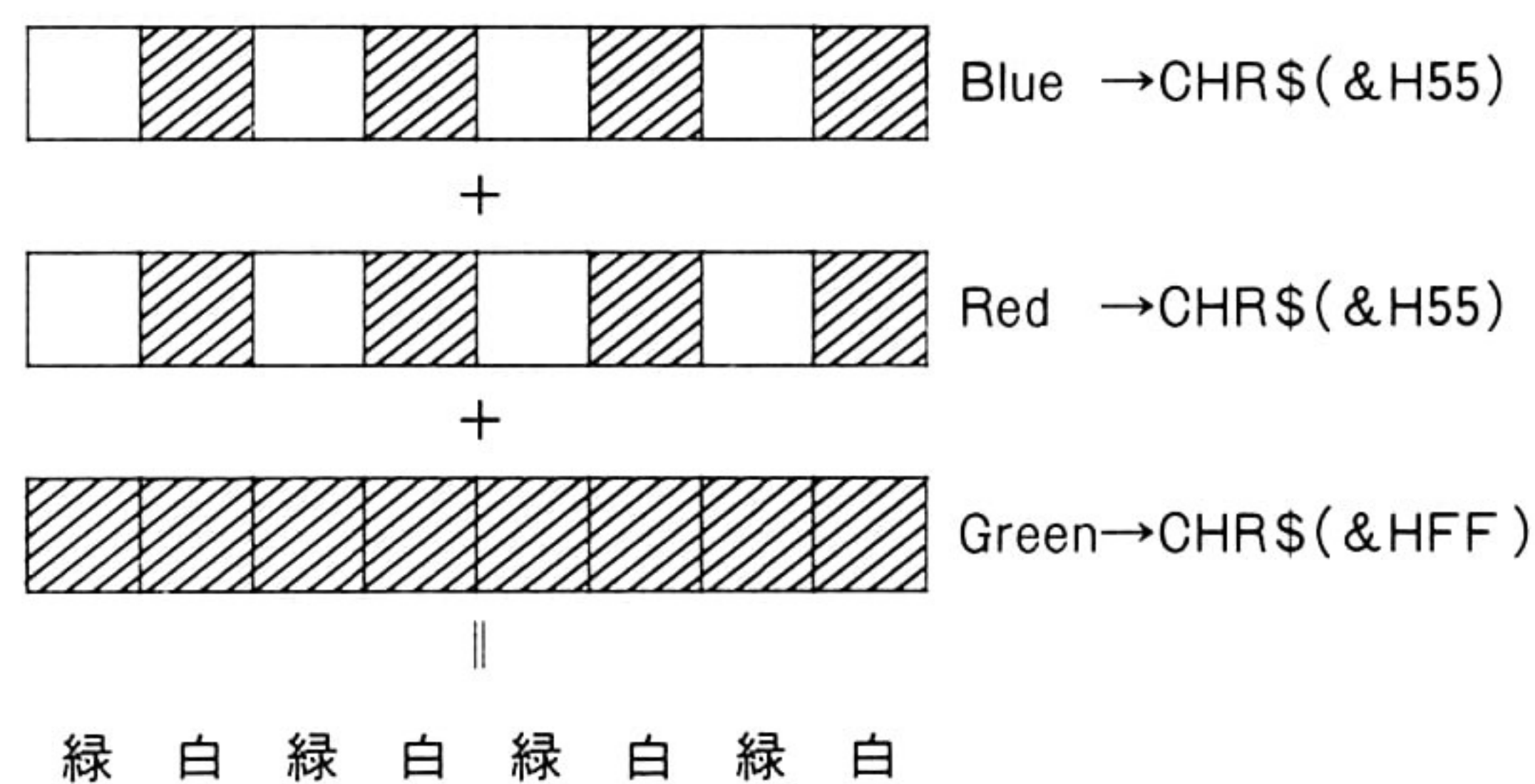



図12 カラーモードにおけるタイル・ストリングの考え方②

ただ注意して欲しいのは，論理的な1ドット内に含まれる実際のドット数，Bn であり，もしこれが大きくなり過ぎると，論理的な1ドットを1ドットと見なすことができなくなります．たとえば，4×4＝16ドットを論理ドットとすると，4913色の色を出せることになりますが，実際は，論理ドット内の1つ1つのドットが個々に見えてしまいます．この論理ドットは，画面のリゾリューション，発色能力，または見る人によって，一概には決まりませんが，PC-8801の場合では，$Bn=2\times2=4$ 位が限度です．しかし，これでも $Cn=(4+1)^3=125$ の色を表現できるのです．

チョット話しが横道にそれてしまいました．

カラーページ参照

```
100 '
110 '  Program 19
120 '    middle color demonstration
130 '
140 '
150 DEF FNROTX(X,Y)=SX*X-SY*Y
160 DEF FNROTY(Y)=SY*P+SX*Y
170  SCREEN 0,0 : CLS 3
180  WINDOW(-300,-300)-(300,300)
190  CIRCLE(0,0),290,7,,,.31
200  CIRCLE(0,0),70,7,,,.31
210  SX=COS(3.1415/6)
220  SY=SIN(3.1415/6)
230  XO=0 : YO=-290
240  XI=0 : YI=-70
250  FOR I=0 TO 11
260    LINE(XO,YO)-(XI,YI),7
270    P=XO
280    XO=FNROTX(XO,YO)
290    YO=FNROTY(YO)
300    P=XI
310    XI=FNROTX(XI,YI)
320    YI=FNROTY(YI)
330  NEXT I
340  LOCATE 34,12 : PRINT "Color  Circle"
350 '
360  TILE$=CHR$(&HFF)+CHR$(0)+CHR$(0)
370  TILE$=TILE$+CHR$(&HFF)+CHR$(0)+CHR$(0)
380 '
390 '  main routine
400 '
410  PX=30 : PY=200
420  RESTORE *TILEDATA
430  FOR I=1 TO 2
440    GOSUB *RED : GOSUB *TILING
450    GOSUB *BLUE : GOSUB *TILING
460  NEXT I
```

```
470  RESTORE *TILEDATA
480  FOR I=1 TO 2
490    GOSUB *GREEN : GOSUB *TILING
500    GOSUB *RED : GOSUB *TILING
510  NEXT I
520  RESTORE *TILEDATA
530  FOR I=1 TO 2
540    GOSUB *BLUE : GOSUB *TILING
550    GOSUB *GREEN : GOSUB *TILING
560  NEXT I
570 END
580 '
590 '  sub routine
600 '
610 *TILING
620  P=PX
630  PX=FNROTX(PX,PY)
640  PY=FNROTY(PY)
650  PAINT (PX,PY),TILE$,7
660 RETURN
670 '
680 *BLUE
690  READ P,Q
700  TILE$=CHR$(P)+MID$(TILE$,2,2)+CHR$(Q)+RIGHT$(TILE$,2)
710 RETURN
720 '
730 *RED
740  READ P,Q
750  TILE$=LEFT$(TILE$,1)+CHR$(P)+MID$(TILE$,3,2)+CHR$(Q)+RIGHT$(TILE$,1)
760 RETURN
770 '
780 *GREEN
790  READ P,Q
800  TILE$=LEFT$(TILE$,2)+CHR$(P)+MID$(TILE$,4,2)+CHR$(Q)
810 RETURN
820 '
830 '  tiling data table
840 '
850 *TILEDATA
860  DATA &H55,&HAA,&HAA,&H55,&HFF,&HFF,&H00,&H00
```

プログラム19は，カラーサークルと名付けてみました．CIRCLE 文の所で説明した，図形を回転させるための行列の計算を行うのが，DEF 文で設定した関数です．360度（2 π）を 6 等分した角度（60度）ずつ回転させる働きをしています．180行の WINDOW 文は，4.3.6の所で述べることなので，ここでは気にせず読み飛ばして下さい．サブルーチンの説明をしておきます．

＊TILING　：塗り始める座標を60度回転移動させ，そこを TILE$で示されるタイル・パターンで埋めつくします．

＊BLUE　：タイル・ストリングの青を示す文字のタイル・パターンを変えます．

＊RED　：タイル・ストリングの赤を示す文字のタイル・パターンを変えます．

＊GREEN　：タイル・ストリングの緑を示す文字のタイル・パターンを変えます．

＊BLUE, ＊RED, ＊GREENの各ルーチンからもわかるように，文字列を操作する関数で処理できます．

最後に，もう 1 つ見慣れないパラメータの，バック・グラウンド・ストリングというのがあります．これは，タイリングを行う領域に既に描かれている色，またはタイル・パターンを指定します．この指定は省略することが可能ですが，そのときは，バック・グラウンド・カラーに対応するバック・グラウンド・ストリングを用います．例えば，バック・グラウンド・ストリングが省略さ

れた時に，COLOR 文で設定されているバック・グラウンド・カラーの色が，緑であったとします．そうすると，省略されたバック・グラウンド・ストリングは，

CHR$ (0)+CHR$ (0)+CHR$ (&HFF)

と見なされます．

しかし，何でも省略してよいとは，限りません．Example 1 の240行では，バック・グラウンド・ストリングを省略しています（したがって，220行は，無駄な行です）．240行の PAINT 文が，既にタイリングされている所を，さらにタイリングするものですが，この例の場合，240行は実行されます．同様に Example 2 の240行も，バック・グラウンド・ストリングを省略しています．しかし，190行は，Example 1 のように TILE$に新しいタイル・ストリングを代入するだけでなく，190行で TILE$ に代入してあったタイル・ストリングに，さらに新しいタイル・ストリングを加えて TILE$ に代入しています．この例は，240行で省略されたバック・グラウンド・ストリングに，バック・グラウンド・カラーに対応したものが，採用されたと見なされますが，240行は実行されません（且つ，エラーの対象にもならない）．

```
100 '
110 '   Example 1
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(0)+CHR$(0)+CHR$(&H55)
200 PAINT(320,100),TILE$,1,STRING$(3,CHR$(0))
210 '
220 BACK$=TILE$
230 TILE$=CHR$(0)+CHR$(&H55)+CHR$(0)
240 PAINT(320,100),TILE$,1
250 END
```

```
100 '
110 '   Example 2
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(0)+CHR$(0)+CHR$(&H55)
200 PAINT(320,100),TILE$,1,STRING$(3,CHR$(0))
210 '
220 BACK$=TILE$
230 TILE$=TILE$+CHR$(0)+CHR$(&H55)+CHR$(0)
240 PAINT(320,100),TILE$,1
250 END
```

では，Example 2 の240行を実行するために，220行で BACK$ に代入していたタイル・ストリングを，バック・グラウンド・ストリングとして240行に書きたしてみましょう．こうすれば，240行も実行できるようになるはずです．

```
100 '
110 '   Example 3
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(0)+CHR$(0)+CHR$(&H55)
200 PAINT(320,100),TILE$,1,STRING$(3,CHR$(0))
210 '
220 BACK$=TILE$
230 TILE$=TILE$+CHR$(0)+CHR$(&H55)+CHR$(0)
240 PAINT(320,100),TILE$,1,BACK$
250 END
```

今度は逆に，常にバック・グラウンド・ストリングを指定する場合に，どうなるかを考えてみます．タイリング時に，バック・グラウンド・ストリングが示す8ドットのパターンが，タイル・ストリングの示すパターンの中に3回以上連続して現われる場合には，Illegal function call エラー（?FC Error）になります．この事柄に注意して，Example 4 を考えてみましょう．200行でTILE$（タイル・ストリング）とBACK$（バック・グラウンド・ストリング）の内容が，全く同じものになっています．プログラムで見た限りは，バック・グラウンド・ストリングの示す8ドットのパターンがタイル・ストリングの示すパターンの中に1回しか現われていないように思えます．そして200行のPAINT文は（補助的な情報である）BACK$を参照しながら，TILE$に従ってタイリングを行おうとします．この場合のTILE$は，8ドット分のタイル・パターンしか示していませんから，195行に，

TILE$=TILE$+TILE$+TILE$

を，入れて実行してもタイル・パターンには，全く影響がありません．したがって，N_{88}-BASICは，タイル・ストリングのパターンの中に，バック・グラウンド・ストリングの示すパターンが，3回以上あると見なすので，Example 4 の場合，210行のPAINT文は，エラーの対象となり，Illegal function call エラーを起こします（実行して，エラーが起こった後，HELP を押せば，PAINT文でエラーが起きたことを確認できます）．このタイル・パターン（TILE$）で，どうしても埋めたいときは，Example 5 のようにバック・グラウンド・ストリング（BACK$）を省略すると，うまく実行できます．当然，Example 5 のようにしたときのバック・グラウンド・ストリングは，バック・グラウンド・カラーに対応したものです．

この場合，もう1つエラーを起こさせない工夫として，Example 6 のように，195行の命令を加える方法があります．この方法は，実質的にTILE$の内容が変わったことになりますが，画面で見た限りでは，ドットの色の組み合わせが同じですから，問題はありません．

```
100 '
110 '   Example 4
120 '
130 '
140 SCREEN 0,1
150 CLS 3
```

```
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1,BACK$
220 END
```

```
100 '
110 '   Example 5
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1
220 END
```

```
100 '
110 '   Example 6
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
195 TILE$=TILE$+CHR$(&H55)+CHR$(&HAA)+CHR$(&HFF)
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1,BACK$
220 END
```

バック・グラウンド・ストリングについても，もう少し説明を加えておきます．バック・グラウンド・ストリングも，タイル・ストリングと同様に，白黒モードで1文字，カラーモードで3文字を必要としますが，異なる点は，それ以上の文字数のときは無視されることです．今まで，話したように，バック・グラウンド・ストリングは，付けて良い場合と，不都合な場合がありますし，付けない方が都合のいい場合もあります．この様々な場合の中には，エラーが起きてもその理由が分かりにくいものも含まれています．例えば，バック・グラウンド・カラーが黒に設定されているとき，

```
TILE$=STRING$(3, CHR$(0))
PAINT(X, Y), TILE$, 7
```

を実行すればエラーが起こります．これは，タイル・ストリングとバック・グラウンド・ストリング（バック・グラウンド・カラーが黒だから）が一致してしまい，Example 4 と同様なことが起っていると考えられます．このように，TILE$の内容がはっきりプログラムに示されているものは，すぐに気付きますが，乱数やある種の計算でタイル・パターンを作った時などには，割と気付きにくいものです．

カラーページ参照

```
100 '
110 '   Program 20
120 '     Tiling demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  GOSUB *COLORINIT
170  POINT(0,190)
180  FOR I=1 TO 10
190    LINE STEP(0,-190)-STEP(63,190),7,B
200  NEXT
210  POINT(630,0)
220  FOR I=1 TO 10
230    LINE STEP(-630,0)-STEP(630,19),7,B
240  NEXT
250 '
260  FOR Y=0 TO 9
270    FOR X=0 TO 9
280      P=X+Y*10
290      BLUE=COL(P,1)
300      RED=COL(P,2)
310      GREEN=COL(P,3)
320      TILE$=""
330      FOR I=0 TO 1
340        TILE$=TILE$+CHR$(TL(BLUE,I))+CHR$(TL(RED,I))+CHR$(TL(GREEN,I))
350      NEXT
360      IF TILE$<>STRING$(6,CHR$(0)) THEN BACK$=STRING$(3,CHR$(0))
                                     ELSE BACK$=STRING$(3,CHR$(&HFF))
370      PAINT(X*63+32,Y*19+10),TILE$,7,BACK$
380    NEXT
390  NEXT
400 END
410 *COLORINIT
420  GOSUB *TILEINIT
430  DIM COL(102,3)
440  NUM=1:ST=0
450  FOR ED=1 TO 3
460    GOSUB *COLINIT
470  NEXT
480  FOR ST=0 TO 3
490    GOSUB *COLINIT
500  NEXT
510  FOR BLUE=0 TO 4
520    RED=BLUE:GREEN=RED
530    GOSUB *CINTSUB
540  NEXT
550 RETURN
560 '
570 *COLINIT
580  BLUE=ST:RED=ED
590  FOR GREEN=ST TO ED-1
600    GOSUB *CINTSUB
610  NEXT
620  FOR RED=ED TO ST+1 STEP -1
630    GOSUB *CINTSUB
640  NEXT
650  FOR BLUE=ST TO ED-1
660    GOSUB *CINTSUB
670  NEXT
680  FOR GREEN=ED TO ST+1 STEP -1
690    GOSUB *CINTSUB
700  NEXT
710  FOR RED=ST TO ED-1
720    GOSUB *CINTSUB
730  NEXT
740  FOR BLUE=ED TO ST+1 STEP -1
750    GOSUB *CINTSUB
760  NEXT
770 RETURN
780 '
790 *CINTSUB
800  COL(NUM,1)=BLUE
810  COL(NUM,2)=RED
820  COL(NUM,3)=GREEN
```

```
830  NUM=NUM+1
840 RETURN
850 '
860 *TILEINIT
870  DIM TL(4,1)
880  RESTORE *TILEDATA
890  FOR I=0 TO 4
900    READ TL(I,0),TL(I,1)
910  NEXT
920 RETURN
930 '
940 *TILEDATA
950  DATA 0,0,136,34,170,85,238,187,255,255
```

プログラム20は，タイル・ストリングとバック・グラウンド・ストリングの内容が，一致しないように，360行でチェックしています．こうすれば，エラーを起こすことなく実行されます．

タイリングの際に，タイル・ストリングとバック・グラウンド・ストリングの関係しか述べませんでしたが，既に画面にある色やタイル・パターンも関係がないとは言えません（バック・グラウンド・ストリングを省略したときに，バック・グラウンド・カラーに対応するストリングが採用されるのも，その例の1つです）．その関係を調べるために，ある実験（と言っても，ほんの試み程度のこと）を，行ってみました．

```
100 '
110 '   Example 7
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
195 TILE$=TILE$+STRING$(3,CHR$(0))
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1,BACK$
220 END
```

まず，Example 7は，途中までしか実行されませんでした．バック・グラウンド・ストリングは，3文字より長い分は，無視されますから，200行で6文字の長さを持つTILE$を代入した所で，BACK$が，210行のPAINT文で有効とするのは，最初の3文字――つまり，

CHR$ (&HAA)+CHR$ (&H55)+CHR$ (&HFF)

の部分のみです．

では，いったい何故途中までしか実行されないのでしょう．バック・グラウンド・カラーは，このとき黒でしたので，150行のCLS命令で画面は黒くなっています．PAINT文はタイル・ストリングの最初の8ドット（つまり3文字）を画面に（横一列に）並べる準備をします．ここで，画面の指定された部分と，この8ドットを比べて，一致しなければ，そのタイル・パターン（8ドット分）を並べていくとします．このパターンが一列並べ終えたら，タイル・ストリングの次の8ドット（3文字）を並べる準備をします．ここで再び，次にタイリングする部分と，この8ドットを比べます．先程は，タイル・ストリングが，CHR$(&HAA)+CHR$(&H55)+CHR$(&

HFF）の部分と画面の黒，すなわちSTRING$（3，CHR$（0））を比較したので，一致していない結果になりましたが，今度の場合に，タイル・ストリングはSTRING$（3，CHR$（0））の8ドットですから，画面と一致しています．バック・グラウンド・ストリングはこのように一致している時に参照され，バック・グラウンド・ストリングの8ドットと画面上のタイリングする部分が，一致していれば，タイリングを行うと考えられます．今は，BACK$は，STRING$（3，CHR$（0））ではないので，タイリングがこの時点で（つまり，途中で）終了してしまいます．

```
100 '
110 '  Example 8
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
195 TILE$=TILE$+STRING$(3,CHR$(0))
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1
220 END
```

以上の考えはBACK$をExample 8のように省略して，バック・グラウンド・ストリングをバック・グラウンド・カラーに対応したものになるようにすると，うまく実行します．これは画面上のタイリングする部分と，タイル・ストリングの後の3文字（8ドット）は一致しても，バック・グラウンド・ストリングで画面（地）のタイル・パターンを指定しているのと同様になるからです．したがって，タイリングが途中で止まったりする場合は，中断が生じた境界のパターンを，バック・グラウンド・ストリングとすることによって中断の解消になることが分かります．もう一つの中断の解消法は，タイル・ストリングのパターンと画面上のパターン（どちらも，8ドット）が，一致しないような場合からタイリングを始める方法です．これは，座標指定を少し変えてやることによって実現できます．

同じように考えれば，Example 9とExample 10の両方とも，タイリングが実行できるのは，理解できるでしょう(Example9,10のBACK$は，STRING$(3,CHR$（0））と見なされます)．

```
100 '
110 '  Example 9
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=STRING$(3,CHR$(0))+CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1,BACK$
220 END
```

```
100 '
110 '   Example 10
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=STRING$(3,CHR$(0))+CHR$(&HAA)+CHR$(&H55)+CHR$(&HFF)
200 BACK$=TILE$
210 PAINT(320,100),TILE$,1
220 END
```

最後に既にタイリングされている領域を，異なるパターンでタイリングしようとした場合，時間がかかったり，スタックを浪費するため，Out of memory エラー（?OM Error）になったりすることがあります．タイリングされている所にさらにタイリングするときは，一度，タイリングする部分を黒で塗りつぶすとか，その部分の枠を違う色で描き直すなどを実行すれば良いと思います．Example 11 とExample 12 は，枠を描き直すとどうなるかを，試したものです．

Example 11 では，250行の PAINT 文は実行されませんが，Example 12 で，225行によって枠を違う色で描き直して，250行の境界色をそれに合わせて変えれば，しっかりと実行します．

タイリングの動作は，様々な状態，条件のもとで実行されますが，エラーを起こしたときに，少しでもこの節で述べたことを役立てて下さい．

```
100 '
110 '   Example 11
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HFF)+CHR$(&H55)+CHR$(0)
200 BACK$=STRING$(3,CHR$(0))
210 PAINT(320,100),TILE$,1,BACK$
220 '
230 BACK$=TILE$
240 TILE$=CHR$(0)+CHR$(&HAA)+CHR$(&H55)
250 PAINT(320,100),TILE$,1,BACK$
260 END
```

```
100 '
110 '   Example 12
120 '
130 '
140 SCREEN 0,1
150 CLS 3
160 WINDOW(0,0)-(639,199)
170 VIEW(250,80)-(390,120),,7
180 LINE(20,10)-(620,190),1,B
190 TILE$=CHR$(&HFF)+CHR$(&H55)+CHR$(0)
200 BACK$=STRING$(3,CHR$(0))
210 PAINT(320,100),TILE$,1,BACK$
220 '
225 LINE(20,10)-(620,190),2,B
230 BACK$=TILE$
240 TILE$=CHR$(0)+CHR$(&HAA)+CHR$(&H55)
250 PAINT(320,100),TILE$,2,BACK$
260 END
```

4.3.5 もう１つのCOLOR(パレット)―色を自由に操る―

今まで，初期状態では，パレット番号とカラーコードは一致している，として話を進めてきましたが，この辺でパレットというものの概念をはっきりしたものにしましょう．まず，グラフィック命令は，色指定の所をパレット番号で指定するということを思い出して下さい．

この『パレット』という単語は，絵を描くときに使われるパレットと同じです．例えば（図13）のようなパレットを考え，そこに左から順に番号をふります．初期状態というのは，このパレットにふった番号と同じカラーコードの絵の具を，同じ番号のパレットの所にしぼり出していると考えて下さい．つまり，パレット０の所に，カラーコード０の色（黒）をしぼり出しているということです．その他のパレットも同様です．例えば，

```
LINE（0，0）－（20，20），2
```

という命令は，座標（0，0）から（20，20）に，パレットの２番（今は，赤の絵の具がしぼり出してある）から，筆で絵の具を取って線を引いている，と解釈できます．

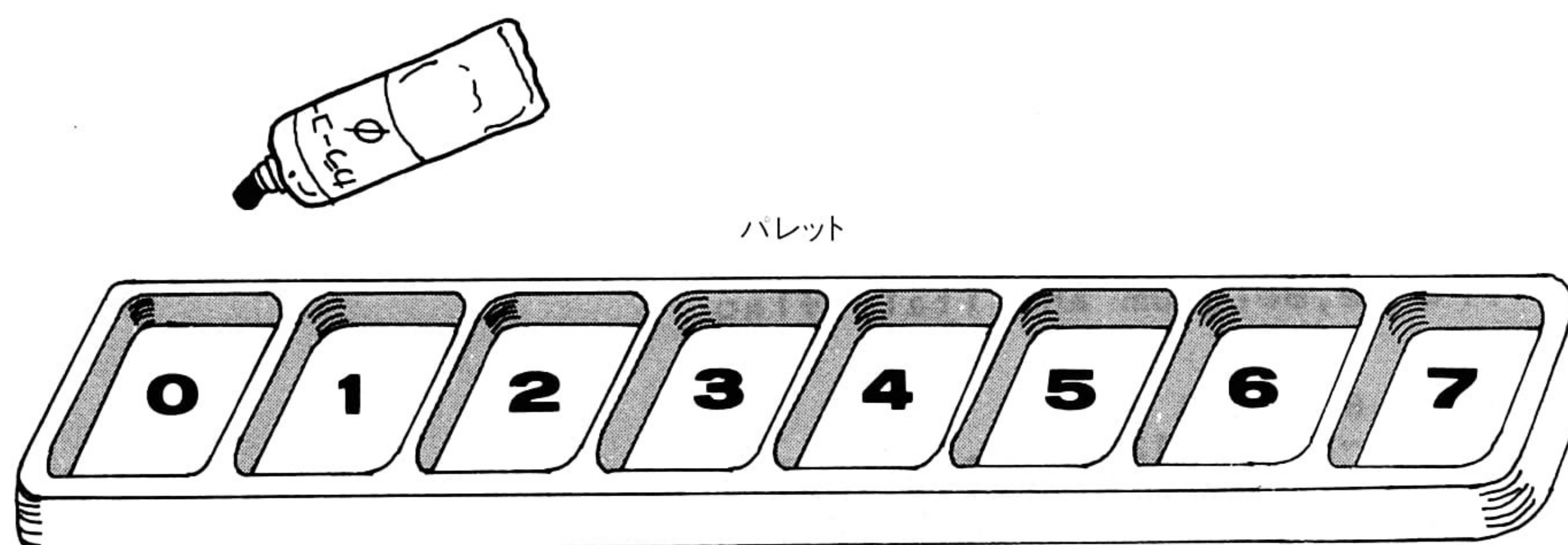



図13　パレットの概念

以上のことは，簡単に理解できるでしょう．では，キャンバス（カンバス）や画用紙などに，絵を描き終ってから，パレットのある場所の絵の具を取り去って，そこに前と異った色の絵の具をしぼり出したら，絵はどうなるでしょうか．当然，絵には何の変化も現われません．しかし，N_{88}-BASICでは，画面に描いた絵の中で，絵の具を取りかえたパレット（番号）の所から色を取って描いた部分は，新しい色に変わります．この様にパレットに乗せてある絵の具の色を変えるのと，同じことをする命令がもう１つのCOLOR文です．書式は，つぎのようにして与えます．

```
COLOR＝（パレット番号，カラーコード）
```

この意味は，『カラーコードで示される番号の色を，パレット番号で示されるパレットに対応させる』です．

まずは，簡単な例を挙げましょう．プログラム21を見て分かるように，190行までは，プログラム７のフランスの国旗を描くプログラムと全く同じです．230行と260行で，例のCOLOR文を使っています．この２つの行で，パレットとカラーコードの結びつきがどうなっていくかを示した

のが（図14）です.

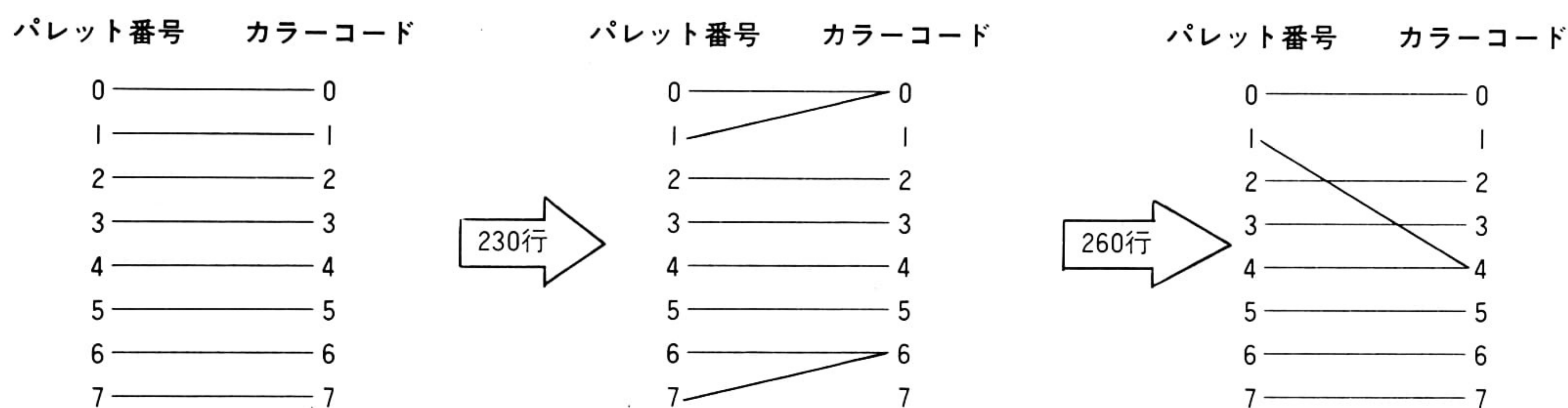
パレット番号　カラーコード　230行→　パレット番号　カラーコード　260行→　パレット番号　カラーコード

プログラム21の230行と，260行のCOLOR文の実行においての，パレットとカラーコードとの結びつきの変化.

図14　パレット番号とカラーコードの対応の変化

いちいち，旗を書き直すことなく一瞬にして，フランスの国旗が，ベルギーの国旗，イタリアの国旗と変わります.

以上のようにパレットを動かした後は290行の様にパレットをきれいに初期状態に戻しておきましょう．パレットを初期に戻しておかないと，カラーコードに何色が指定されているのか，わからなくなってしまいます.

カラーページ参照

```
100 '
110 '  Program 21
120 '    France,Belgium and Italy flag
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  LINE(0 0)-STEP(213,182),1,BF
170  LINE STEP(1,-182)-STEP(213,182),7,BF
180  LINE STEP(1,-182)-STEP(213,182),2,BF
190  LOCATE 0,23 : PRINT "FRANCE FLAG";
200 '
210  IF INKEY$="" THEN 210
220  LOCATE 0,23 : PRINT "BELGIUM FLAG";
230  COLOR=(1,0) : COLOR=(7,6)
240  IF INKEY$="" THEN 240
250  LOCATE 0,23 : PRINT "ITALY FLAG  ";
260  COLOR=(1,4) : COLOR=(7,7)
270  IF INKEY$="" THEN 270
280 '
290  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
300 END
```

COLOR 文で，次に行うのはアニメーションのような手法です．プログラム22とプログラム23にその例を示します．この方法ですと実際，絵は動いていないのですが，パレットとカラーコードの関係を巧みに変えることによって，動いて見えるのです．プログラム22は，PAINT 文で塗った面の色を，200行以降で，ローテーション（回転）させています．プログラム23は，全てのパレット番号を用いて，位相の異なったサインカーブを８つ描きます．そしてパレットを用いることによって，常に画面上には，１つのサインカーブしか現われないようにしながら，波が進んで見えるように，250行でパレットの内容を変えています.

```
100 '
110 '   Program 22
120 '     palette demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3 : C=1
160  FOR R=50 TO 300 STEP 10
170    CIRCLE(320,100),R,C,,,.31
180    C=((C+1) MOD 7)+1
190  NEXT
200 '
210 '   palette rotation
220 '
230  FOR J=1 TO 50
240    CO=1
250   *LOOP
260    FOR I=1 TO 7
270      COLOR=(I,((CO+I) MOD 7)+1)
280    NEXT
290    CO=CO+1
300    IF CO<6 THEN *LOOP
310  NEXT
320 '
330  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
340 END
```

```
100 '
110 '   Program 23
120 '     palette demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  C=1
170  FOR I=0 TO 3.1415*2 STEP 3.1415*2/7
180    FOR X=-3.1415*2 TO 3.1415*4 STEP .05
190      PY=-75*SIN(X)+100 : PX=(X+I)*50
200      PSET(PX,PY),C
210    NEXT
220    C=C+1
230  NEXT
240 '
250  FOR I=1 TO 7 : COLOR=(I,0) : NEXT
260  FOR T=0 TO 10
270    FOR J=1 TO 7
280      FOR I=1 TO 7
290        COLOR=(I,J)
300        FOR D=0 TO 50 : NEXT
310        COLOR=(I,0)
320      NEXT
330    NEXT
340  NEXT
350  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
360 END
```

カラーページ参照

単純なアニメしかできませんが，パレットの概念というのは，工夫すれば意外に面白い利用法がアニメ以外にも見つかりますから，是非、挑戦してみて下さい．例えば，PAINT 文で枠を描かなければなりませんが，この枠を COLOR 命令で見えないようにしておいて，PAINT することもその一つでしょう．

4.3.6 WINDOW(ワールド座標とは)—座標を自由に設定する—

N_{88}-BASIC の偉大な機能の１つである，パレットの概念について述べましたが，さらに幾つかの偉大な（要するに，本格的なグラフィックの考え方を備えているということです）機能について，紹介したいと思います．

dot数＼モード	カラーモード	白黒モード
640×200	(0, 0)→(639, 0) (0, 199) (639, 199)	(0, 0)→(639, 0) (0, 199) (639, 199)
640×400	存在しない	(0, 0)→(639, 0) (0, 399) (639, 399)

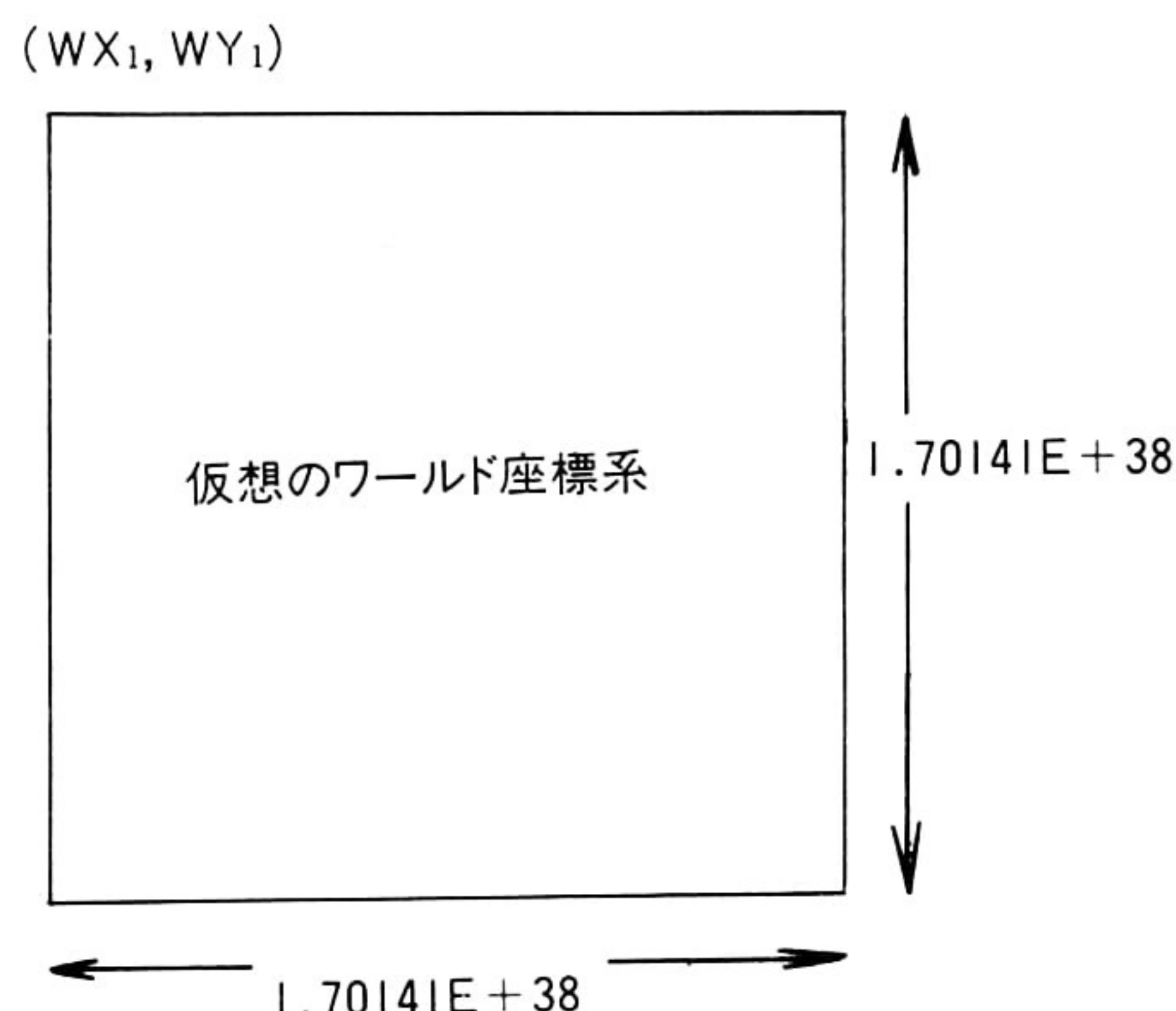
表3の設定によりこのような座標になる．

表12　モードとdot数

これまではグラフィック座標として，640×200，もしくは640×400として画面と座標を結びつけてきました．つまり，640×200のときは(表12)のように座標系を考えてきた訳です．しかし，実際には，今まで紹介してきたプログラムの動作は，(表12)のようなグラフィック座標上で実行されたのではなく，これから説明するワールド座標という座標上で，実行されていたのです．この座標は，N_{88}-BASICが持っている仮想の座標で(図15)の様な大きさを持っています．つまり，グラフィック座標では，ワールド座標上の一部分をながめていたという訳です．

(WX1, WY1)

仮想のワールド座標系

1.70141E＋38

1.70141E＋38

ワールド座標上の座標を示す数値は，縦横とも，
−1, 70141E＋38〜＋1, 70141E＋38の範囲で許されている．

図15　ワールド座標系における座標の範囲

(−500, −500)

WINDOW

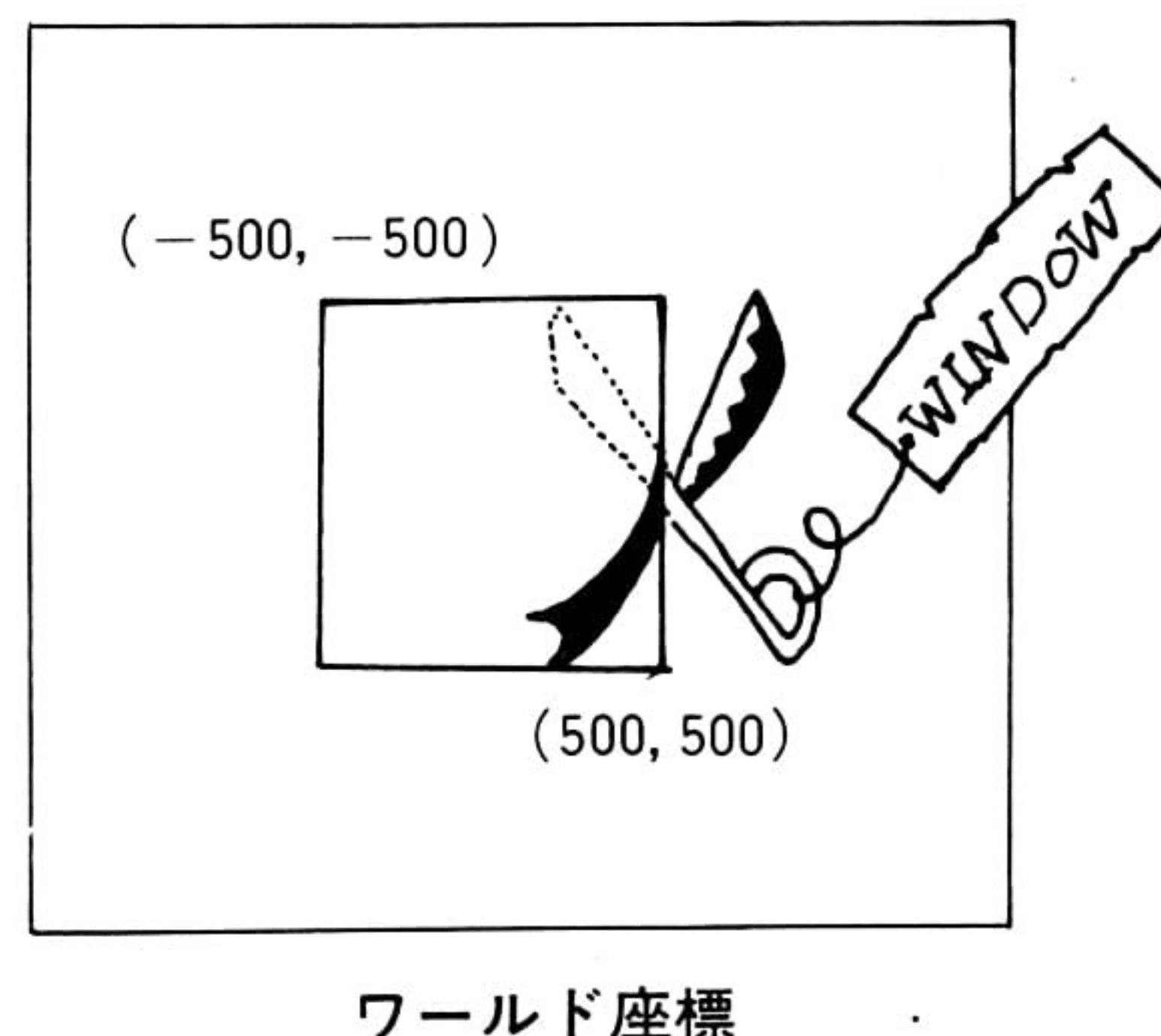
(500, 500)

ワールド座標

図16　WINDOWとは，ワールド座標を切りとるはさみ?!

では，この広い座標をどう使ったらよいのでしょうか？　実際，必要なのでしょうか？　これらの質問とは別にして，WINDOW 命令とはどういうものなのか，定義してみましょう．

『WINDOW とは，文字通り窓のようなもので，ワールド座標（という紙）の任意の（自分の好きな）部分を切りとる，はさみのような道具（命令）であり，この切りとった部分は，画面の大きさに合わせて，拡大または縮小されて表示される』

例えば，

WINDOW (−500, −500)− (500, 500)

と書けば（図16）で示される部分を切りとることであり，この切りとった部分をカラーモードなら 640×200 の大きさの画面の大きさに合わせて表示させます．但し，WINDOW 命令は，既に描かれている絵や図には何の影響も与えません．

プログラム24は，質問の１つ目『どう使ったらよいのか？』ということに答えるために，一例として考えてみました．この折れ線グラフは，東京における 1941年から 1970年までの平均の降水量を示したもので，一年間を５日単位で表示しています．一年間の表示が終ると次に，どの期間を細かく見たいのかを促してきます．ここでいう期間というのは，１年間を25日単位で区切ったもので，(表13)を見て入力してみて下さい．そうすると，その指定した期間のグラフだけが細かく表示されます．これは，折れ線グラフを描く手法は同じですが，表示される部分は，WINDOW 文で異なった範囲になっています．『どう使ったらよいか？』ということが多少なりとも理解して頂けたでしょうか．

コード	期　間	コード	期　間
1	1/1 ～ 1/25	8	6/25 ～ 7/19
2	1/26 ～ 2/19	9	7/20 ～ 8/13
3	2/20 ～ 3/16	10	8/14 ～ 9/7
4	3/17 ～ 4/10	11	9/8 ～ 10/2
5	4/11 ～ 5/5	12	10/3 ～ 10/27
6	5/6 ～ 5/30	13	10/28 ～ 11/21
7	5/31 ～ 6/24	14	11/22 ～ 12/16
		15	12/27 ～ 12/31

表13　プログム24で参照する表

```
100 '
110 '  Program 24                                    カラーページ参照
120 '     window demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  WINDOW(-30,-560)-(730,40)
170 '
180 *START
190  CLS 2
200 '
210  LINE(0,-510)-(0,0),7
220  LINE -STEP(730,0),7
230  FOR I=0 TO -500 STEP -50
240    LINE(-5,I)-(0,I),7
250  NEXT
260  FOR I=0 TO 700 STEP 50
270   LINE(I,-10)-(I,0),7
280  NEXT
290  GOSUB *NUMBER
300 '
310  RESTORE *PRECIPITATION
320  READ A$,P : X=0 : POINT(10,-P)
330 '
340  RESTORE *PRECIPITATION
350  FOR I=1 TO 15
360    READ A$
370    FOR J=1 TO 5
380      READ P : X=X+10
390      LINE -(X,-P),5
400      CIRCLE STEP(0,0),1,3
410    NEXT
420  NEXT
430  LOCATE 0,0 : PRINT TAB(20);"*ﾍｲｷﾝ ｺｳｽｲ ﾘｮｳ (5B ｵｷ)"
440  PRINT "(mm)";SPC(20)
450  IF INKEY$="" THEN 450
460 '
470  LOCATE 0,0 : PRINT SPC(70)
480  LOCATE 0,0 : INPUT "ﾄﾞﾉ ｷｶﾝ ｦ､ ｺﾏｶｸ ﾐﾀｲ ﾃﾞｽｶ ";N
490  IF N=0 OR N>15 THEN END
500  ON N GOSUB *SET1,*SET2,*SET3,*SET4,*SET5,*SET6,*SET7,*SET8,
                *SET9,*SET10,*SET11,*SET12,*SET13,*SET14,*SET15
510  LINE((N-1)*50+10,-510)-(N*50,0),2,B
520  INPUT "ｺﾉ ﾊﾝｲ ﾃﾞ ｲｲﾃﾞｽｶ ";C$
530  IF C$<>"y" THEN *START
540  CLS 3
550 '
560  READ A$ : LOCATE 20,0
570  PRINT A$;" ﾉ ﾍｲｷﾝ ｺｳｽｲﾘｮｳ"
580  PRINT "(mm)"
590  WINDOW(N*50-10,-560)-(N*50+60,40)
600 '
610  LINE(N*50-5,-510)-(N*50-5,0),7
620  LINE -STEP(50,0),7
630  FOR I=0 TO -500 STEP -50
640    LINE(N*50-5.5,I)-(N*50-5,I),7
650  NEXT
660  FOR I=N*50 TO N*50+40 STEP 10
670    LINE(I,-10)-(I,0),7
680  NEXT
690  GOSUB *NUMBER
700 '
710  READ P : X=N*50 : POINT(X,-P) : CIRCLE STEP(0,0),.2,2
720 '
730  FOR I=2 TO 5
740    READ P : X=X+10
750    LINE -(X,-P),1
760    CIRCLE STEP(0,0),.2,2
770  NEXT
780  IF INKEY$="" THEN 780
790 END
800 '
810 *NUMBER
820  LOCATE 0,2 : PRINT " 50"
```

```
830  LOCATE 0,12 : PRINT " 25"
840  LOCATE 0,23 : PRINT "  0";
850 RETURN
860 '
1000 *SET1  RESTORE *DATA1  : RETURN
1010 *SET2  RESTORE *DATA2  : RETURN
1020 *SET3  RESTORE *DATA3  : RETURN
1030 *SET4  RESTORE *DATA4  : RETURN
1040 *SET5  RESTORE *DATA5  : RETURN
1050 *SET6  RESTORE *DATA6  : RETURN
1060 *SET7  RESTORE *DATA7  : RETURN
1070 *SET8  RESTORE *DATA8  : RETURN
1080 *SET9  RESTORE *DATA9  : RETURN
1090 *SET10 RESTORE *DATA10 : RETURN
1100 *SET11 RESTORE *DATA11 : RETURN
1110 *SET12 RESTORE *DATA12 : RETURN
1120 *SET13 RESTORE *DATA13 : RETURN
1130 *SET14 RESTORE *DATA14 : RETURN
1140 *SET15 RESTORE *DATA15 : RETURN
1150 '
1160 '
2000 *PRECIPITATION
2010 '
2020 *DATA1  :DATA    1/1-1/25,116, 52, 63, 54, 86
2030 *DATA2  :DATA   1/26-2/19, 75,127,107, 92,115
2040 *DATA3  :DATA   2/20-3/16, 91,102,130,132,160
2050 *DATA4  :DATA   3/17-4/10,172,167,185,179,219
2060 *DATA5  :DATA    4/11-5/5,216,201,203,201,246
2070 *DATA6  :DATA    5/6-5/30,189,235,310,245,213
2080 *DATA7  :DATA   5/31-6/24,251,235,344,348,289
2090 *DATA8  :DATA   6/25-7/19,386,254,270,244,177
2100 *DATA9  :DATA   7/20-8/13,290,150,135,109,172
2110 *DATA10 :DATA    8/14-9/7,179,316,446,315,109
2120 *DATA11 :DATA    9/8-10/2,207,405,320,482,306
2130 *DATA12 :DATA  10/3-10/27,498,419,146,229,256
2140 *DATA13 :DATA 10/28-11/21,281,191, 61,135,227
2150 *DATA14 :DATA 11/22-12/16,187,194,125, 69, 96
2160 *DATA15 :DATA 12/17-12/31, 86, 90, 72,  0,  0
```

質問の２つ目『実際，必要であるのか』ということに答えてみましょう．何度か話の中に出て来た，原点を中心とする回転移動を用いて，正多角形を描くのがプログラム25です．これが何故，質問に答えていることになるのでしょう．それは，正多角形を描くのに，原点を中心とした回転移動を用いているからです．この原点（0，0）を，WINDOW 文は画面の中心に移しています．もし，WINDOW 文が使えないとしたら，計算のときに，原点があたかも画面の中心にあるように考えなければなりません．たいして面倒なことではないのですが，WINDOW 文があれば，やはり，便利であることは確かです．

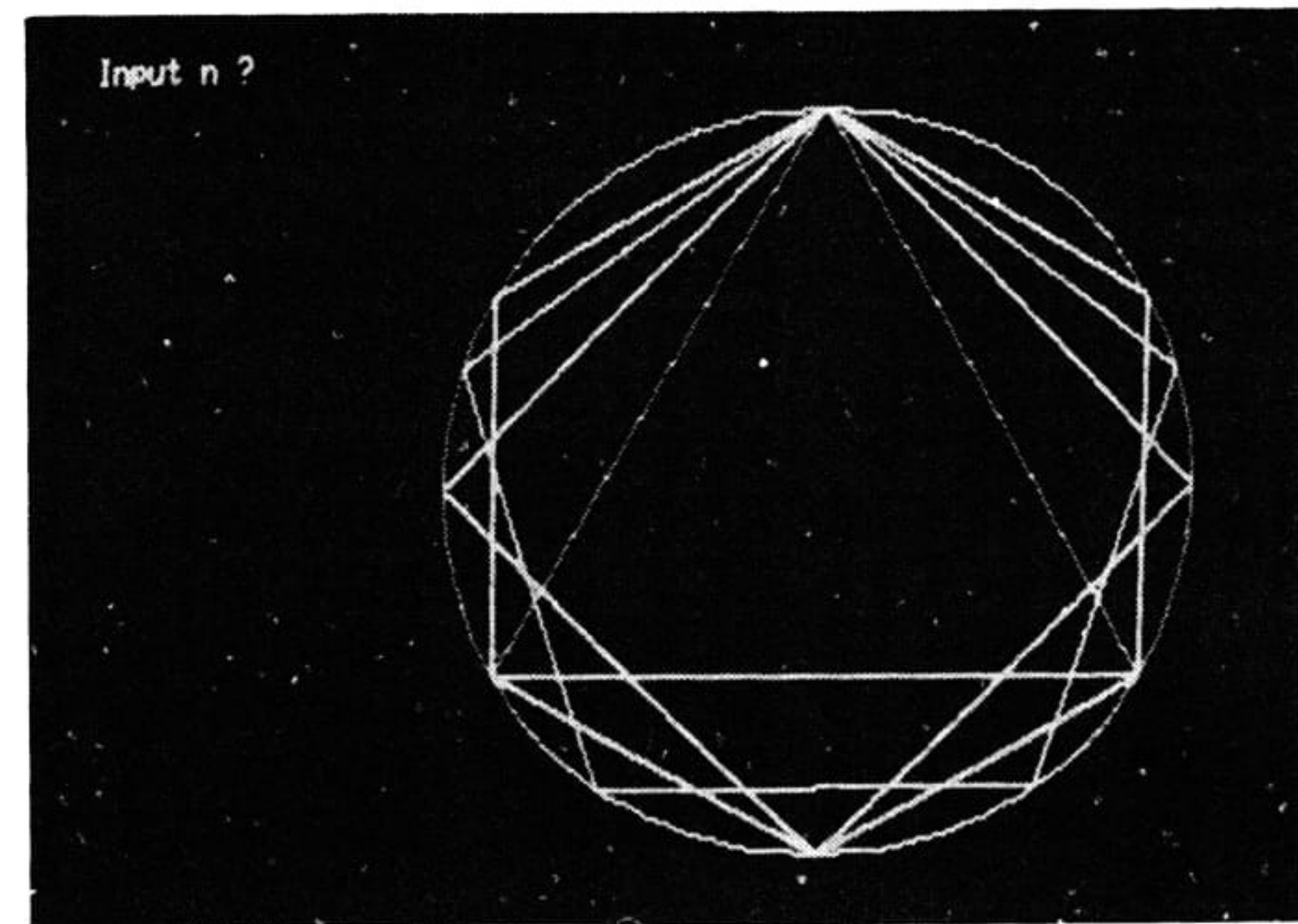
ここで述べたことは，ほんの一例に過ぎませんから，豊かな発想で他の利用法も考えてみて下さい．例えば，世界地図の略図を作るときなどに，WINDOW の切り取り方を

WINDOW (−180, −90)− (180, 90)

とすれば，緯度と経度をデータにするとき，画面の大きさを考えずにそのままの緯度と経度を入力してやるだけでいいのです（但し，この場合は，北緯と西経を負，南緯と東緯を正，などとして与える必要があります．また，二点間を結ぶときに，東回りで結ぶのか，西回りで結ぶのかという情報も必要になってきます．この情報をなくすには，相対座標を用いるのがよいでしょう）．

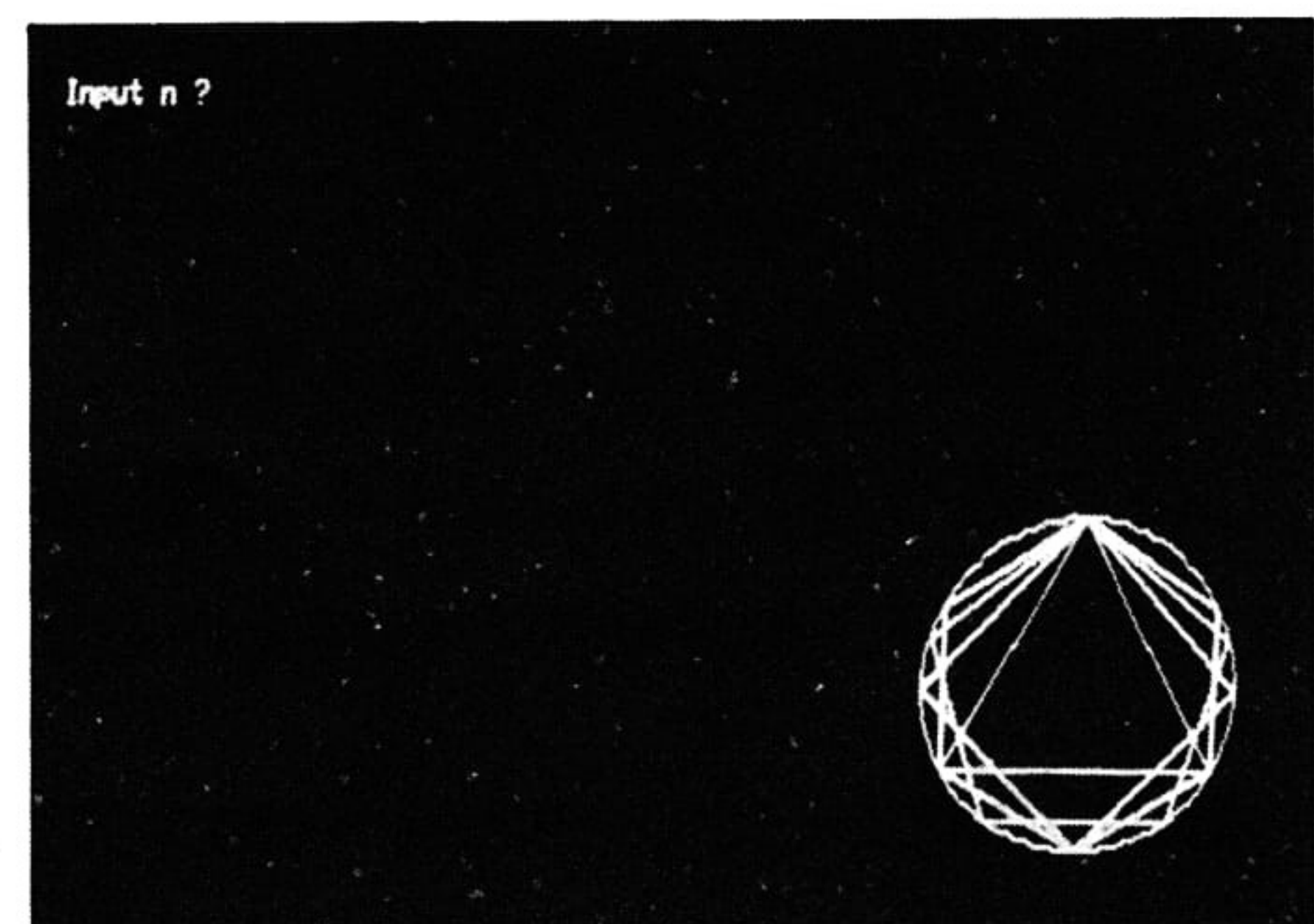
WINDOW と次の節で述べる VIEW は，図形の拡大，縮小に用いられるのが，代表的です．

プログラム 24も，結局，拡大を利用したものです．また，プログラム 25も，WINDOW を大きくとれば図は縮小されるのです．

```
100 '
110 '  Program 25
120 '    セイ タカッケイ ノ demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  WINDOW(-600,-350)-(600,350)
170 *AGAIN : CLS
180  INPUT "Input n ";N
190  IF N=0 THEN *FIN
200  K=2*3.1415/N : C=(N MOD 7)+1
210  SX=COS(K)  : SY=SIN(K)
220  X=0 : Y=-300 : POINT(X,Y)
230  FOR I=1 TO N
240    P=X
250    X=SX*X-SY*Y
260    Y=SY*P+SX*Y
270    LINE -(X,Y),C
280  NEXT : GOTO *AGAIN
290 *FIN : END
```

4.3.7 VIEW(スクリーン座標とは)—表示画面の大きさを自由に設定する—

WINDOW 命令が，ワールド座標を切りとったのに対して，VIEW 命令は画面上の座標を切りとるための道具です．そして，VIEW で切りとられた範囲の中に存在するのが，スクリーン座標です．（図17）は，WINDOW とVIEW の関係を示しています．WINDOW 文で切りとられた範囲は，画面上の VIEW 文で区切られた範囲の大きさに合わせて表示されます．VIEW 命令は一口で言ってしまえば，画面上のどこに表示するかを設定するものに他なりません．プログラム25に次の文を付け加えてみたのがプログラム26です．

```
175  VIEW (339,99)— (639, 199)
```

```
100 '
110 '  Program 26
120 '    セイ タカッケイ ノ demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  WINDOW(-600,-350)-(600,350)
170 *AGAIN : CLS
175  VIEW(339,99)-(639,199)
180  INPUT "Input n ";N
190  IF N=0 THEN *FIN
200  K=2*3.1415/N : C=(N MOD 7)+1
210  SX=COS(K)  : SY=SIN(K)
220  X=0 : Y=-300 : POINT(X,Y)
230  FOR I=1 TO N
240    P=X
250    X=SX*X-SY*Y
260    Y=SY*P+SX*Y
270    LINE -(X,Y),C
280  NEXT : GOTO *AGAIN
290 *FIN : END
```

何をしたのかは，実行すればすぐに分かります．この場合は，(339.99) と (639, 199) を対角線とする，長方形の中で描いていた絵を縮小しています．このとき WINDOW と VIEW の関係

を示すのが，（図18）です．このVIEWを利用すれば，ある図形を描くプログラムを1つ用意しておいて，図を描く度にVIEW命令を実行すれば，たくさんの同じ形の図を好きな位置に描くことができる訳です．プログラム27は，その一例です．乱数によって，VIEWの位置をでたらめに決めています．＊STARDRAWは星を描くサブルーチンでしつこいようですが，回転の行列を使っています．

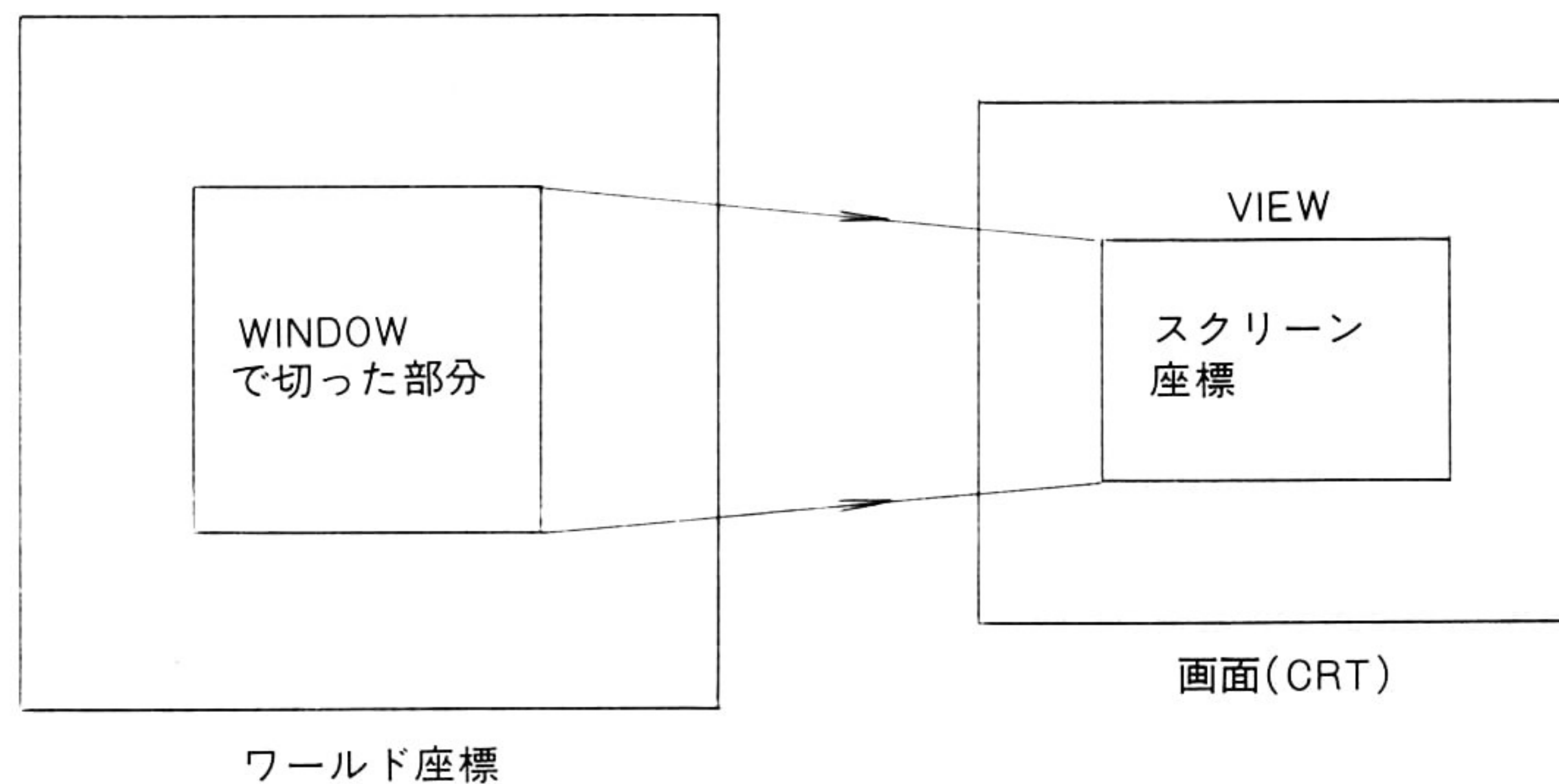



図17　WINDOWとVIEWの関係(1)

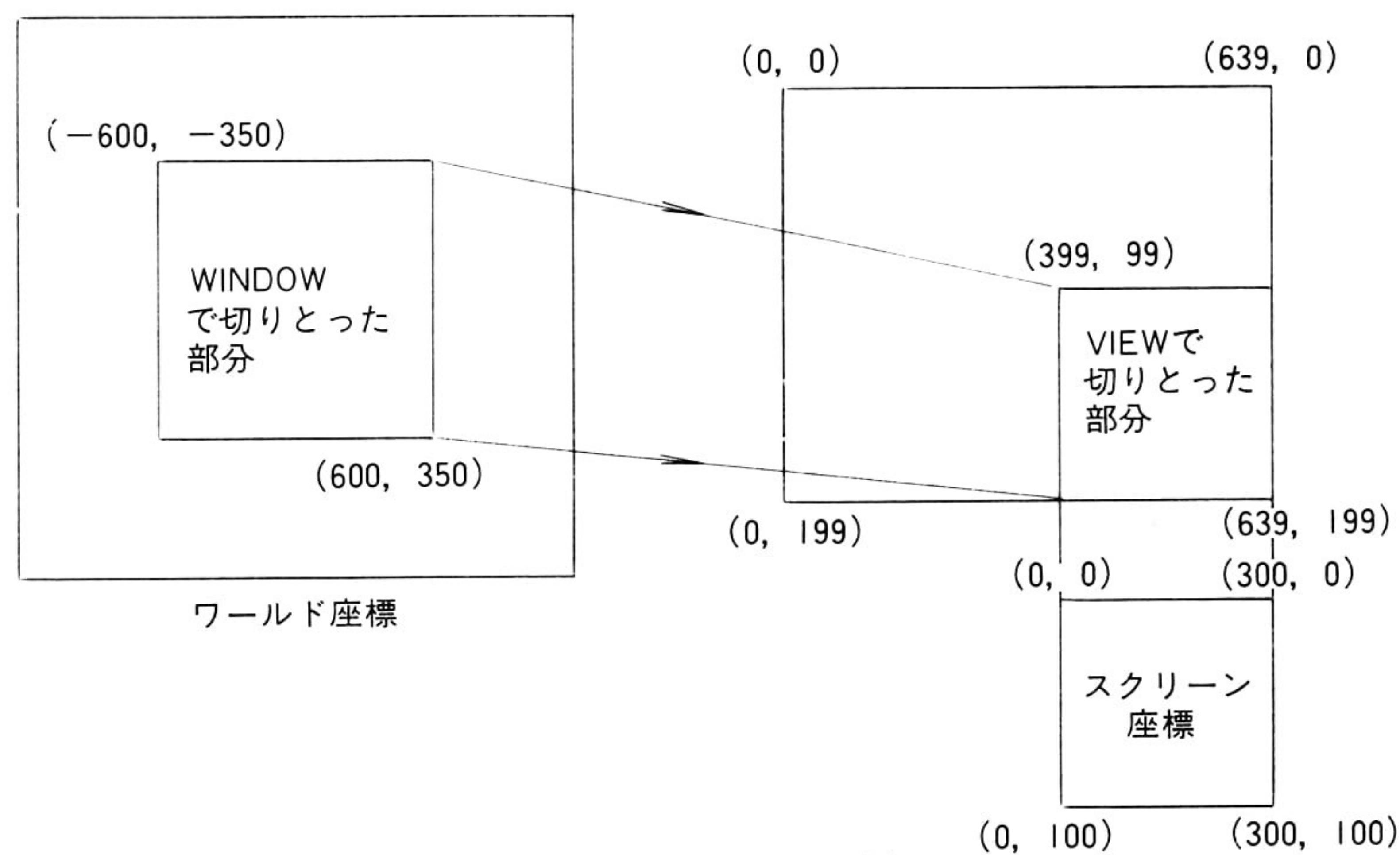



図18　WINDOWとVIEWの関係(2)

カラーページ参照

```
100 '
110 '   Program 27
120 '     the Stars drawing demonstration
130 '
140 '
150  SCREEN 0,3 : CLS 3
160  SCREEN ,0 : WINDOW(-300,-300)-(300,300)
170  RANDOMIZE : CLS : C=1
180  FOR L=1 TO 10
190    VX=RND(1)*600 : VY=RND(1)*160
200    VIEW(VX,VY)-(VX+39,VY+19)
210    GOSUB *STARDRAW
220    C=(C MOD 7)+1
230  NEXT
240 END
250 '
260 '   subroutine (star pattern draw)
270 '
```

```
280 'STARDRAW
290  K=2*3.1415/5
300  SX=COS(K) : SY=SIN(K)
310  X=0 : Y=-290 : POINT(X,Y)
320  FOR I=1 TO 5
330    FOR J=1 TO 2
340      P=X : X=SX*X-SY*Y
350      Y=SY*P+SX*Y
360    NEXT J
370    LINE -(X,Y),C
380  NEXT I
390 '
400  PAINT(0,0),C,C
410  X=0 : Y=-140
420  FOR I=1 TO 5
430    P=X : X=SX*X-SY*Y
440    Y=SY*P+SX*Y
450    PAINT(X,Y),C,C
460  NEXT I
470 RETURN
```

特に，VIEW 文で設定した枠の中をビューポートと呼びます．ビューポートには，前に述べた通り，スクリーン座標というものが存在します．この座標の大きさは，VIEW 命令でパラメータの設定の仕方に左右されます．座標の大きさというのは，横方向の座標と縦方向の座標が取れる最大値のことで，先程の例，

VIEW (339, 99)－(639, 199)

では，それぞれ座標の最大値は次のようにして求めます．

横方向：639－339＝300

縦方向：199－99＝100

したがって，スクリーン座標は，(図18）のようになる訳です．

グラフィック命令のほとんどは，4.3.6の WINDOW の所で述べたワールド座標上で行われますが，スクリーン座標を用いる命令も，幾つかあります．その1つが，今まで何の気なしに使っていた CLS命令です．CLS 命令は座標のパラメータを伴いませんから，直接スクリーン座標には関係ありませんが，スクリーン座標が存在しているビューポートに関係しているので，ここに挙げてみました．4.3.1の所で述べた CLS 命令を思い出して下さい．パラメータによって3通りの意味を持たせていましたが，このうちのグラフィック画面を消す命令（パラメータが，2または，3の場合）というのは，実は，画面全体を消すのではなく，ビューポート内のみを消していたのです．これを用いると，画面上の一部分（但し，その形は，長方形に限られますが……）を VIEW 命令で指定しておいて，CLS 命令を実行すれば，ビューポート内のみを消すのですから，結局，画面の一部分を消すことができる訳です．なお VIEW 命令は，WINDOW 命令と同様に実行しても画面上に既に描かれたものに影響を与えませんから，描き終った図などの，一部分を消して描き直すといったこともできます．

VIEW に関して，もう少し説明することがあります．今，VIEW で指定した範囲内を一色で塗

りつぶしたいとしますと，幾つか方法があります．その1つは，バック・グラウンド・カラーを変えて，CLS 命令を実行するものです．次に考えられるのは，LINE 文のBF のオプションを用いるものでしょう．第3の方法は PAINT 文を用いるといった所でしょうか．しかし，どれを取っても，VIEW 命令以外に，幾つか命令を書かなければいけないようです．これを解消するためにVIEW命令の後に，コンマで区切って領域色の指定ができます．例えば，

```
VIEW (0, 0)-(10, 15), 4
```

というのは，ビューポート内をパレット番号4の色で塗りつぶすことを意味します．

では，塗りつぶさずにビューポートに枠だけを付けるには，どうしたら良いでしょう．もちろん，その枠は CLS 命令で消えないものです．LINE 文のBのオプションを用いて，CLS 命令で消えない枠を描くには，ビューポートの範囲を考える必要があります．この作業も簡略化できないでしょうか．

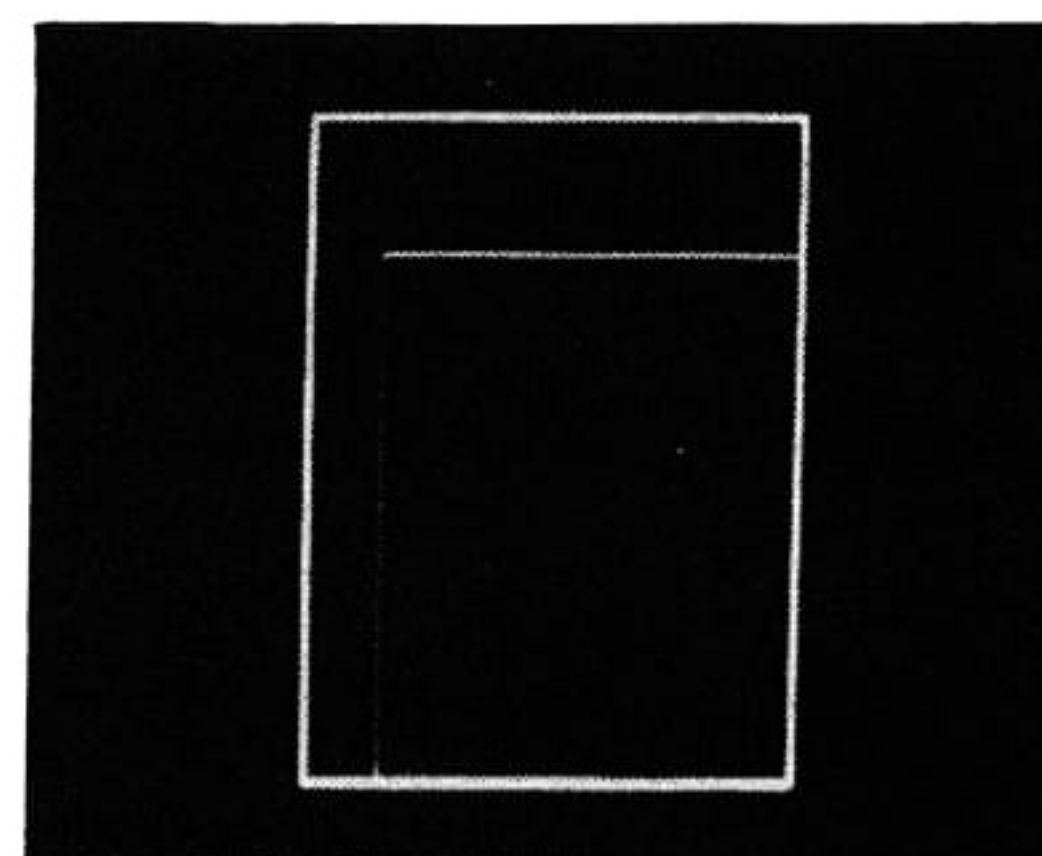
```
VIEW (0, 0)-(10, 15), 4, 1
```

とすれば，枠を LINE 文に頼らず描き，且つ，ビューポート内を塗りつぶします．領域色の後に，コンマで区切って指定してあるのが境界色で，領域色と同様にパレット番号で指定します．塗りつぶしたくないときは，領域色のパラメータを指定しなければ，枠だけ描いてくれます．これらの機能も，知っていれば無駄な手間を省く便利な機能だと言えるでしょう．

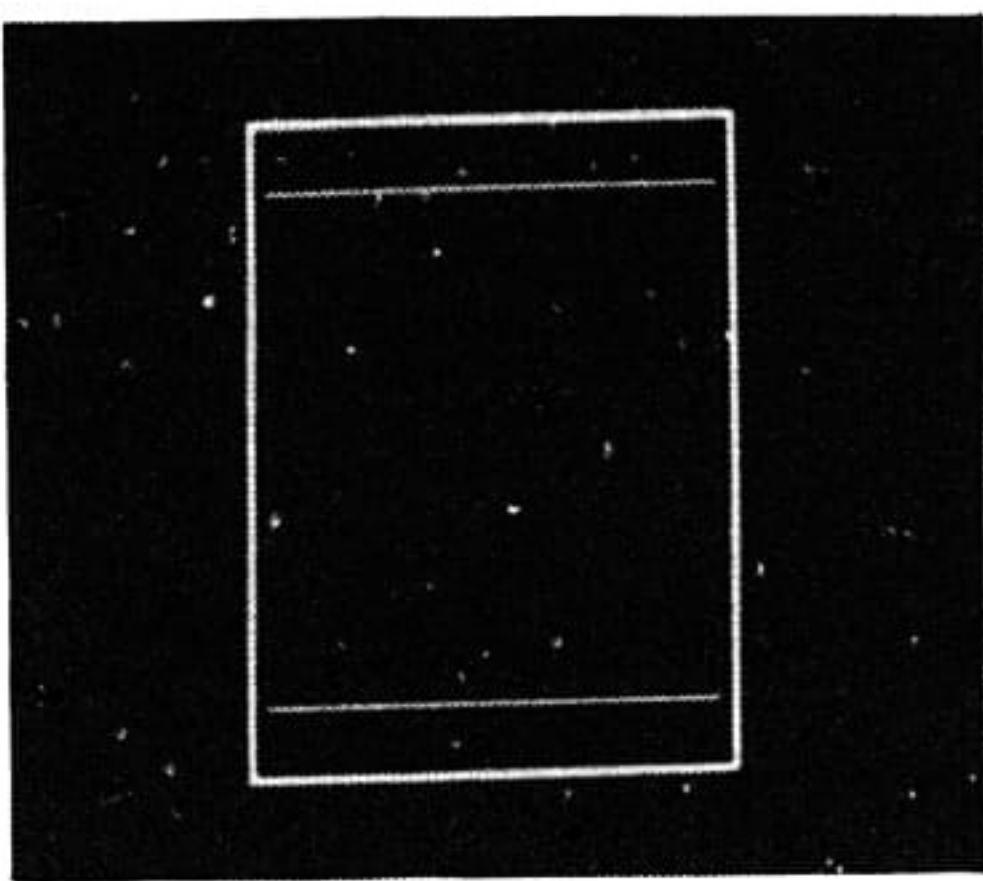
さらに，ここで WINDOW 命令との関係について述べておきたいと思います．プログラムなどでウィンドウやビューポートなどが初期状態（WINDOW 命令や VIEW 命令が実行されていない状態）で，VIEW 命令だけ実行させたとしましょう．これだけで（図17）のような関係になるでしょうか．もし，640×200のモードであったとすると，初期状態のウィンドウのエリアは，

```
WINDOW (0, 0)-(639, 199)
```

を実行したものと同じ範囲です．しかし，このWINDOW 文を実行するのとしないのとでは，VIEW命令を実行したときに異った影響が出てきます．例えば，次のプログラム28を実行させてみて下さい．

```
100 '
110 '   Program 28
120 '     window & view [1]
130 '
140 '
150 SCREEN 0,0 : CLS 3
160 VIEW(250,50)-(390,150),,7
170 LINE(20,20)-(619,179),2,B
180 END
```

前のプログラムは，白い枠の内側に赤い枠の箱を描くつもりで作ったのですが，うまく描けません．全く（図11）の関係が成り立っていないようです．つまりワールド座標のウィンドウが，ビ

ューポート内に反映していないことになります．では，次のように155行をつけ加えたのがプログラム29です．実行してみて下さい．

```
100 '
110 '   Program 29
120 '     window & view [2]
130 '
140 '
150 SCREEN 0,0 : CLS 3
155 WINDOW(0,0)-(639,199)
160 VIEW(250,50)-(390,150),,7
170 LINE(20,20)-(619,179),2,B
180 END
```

白い枠の内側に赤い枠の箱が描けたと思います．

以上のことから，ワールド座標のウィンドウが，ビューポート内に反映するのは，WINDOW 文の実行後のみであることがわかります．WINDOW 文を実行しないで VIEW 命令だけ実行すると，ビューポート内のスクリーン座標とワールド座標の値は，一致したままになっています。(図17）の関係を持たせるためには，必ず VIEW 命令より前に，WINDOW 命令を実行させましょう．

4.3.8 POINT関数，MAP関数—便利な関数その1—

スクリーン座標（ビューポート）に関係した命令として，前の節でふれたのは，CLS 命令と，POINT 関数です．この場合の POINT 命令は，4.3.2で紹介した，LP を設定する POINT 命令とは別のものです．別といっても，全く関係がない訳ではありません．

前に紹介した POINT 文は，LP を設定するものでした．では逆に現在 LPが設定されている座標を知りたいとしたら，どうすればよいのでしょうか？　その様な時に，役立ってくれるのがここで紹介する POINT 関数です．POINT 関数の与える座標の値は，1つだけですから，パラメータによって必要な座標を示す必要があります．(表14）を見て下さい．

パラメータ	機　　能
0	LPのX座標をワールド座標上の値として返す
1	LPのY座標をワールド座標上の値として返す
2	LPのX座標をスクリーン座標上の値として返す
3	LPのY座標をスクリーン座標上の値として返す

表14　POINT関数のパラメータと機能

パラメータに与える数値別に，それぞれの機能（関数が返してくる値）を示しております．そのパラメータを POINT 関数に理解させるためには，次のようにパラメータをカッコで括ります．

POINT（パラメータ）

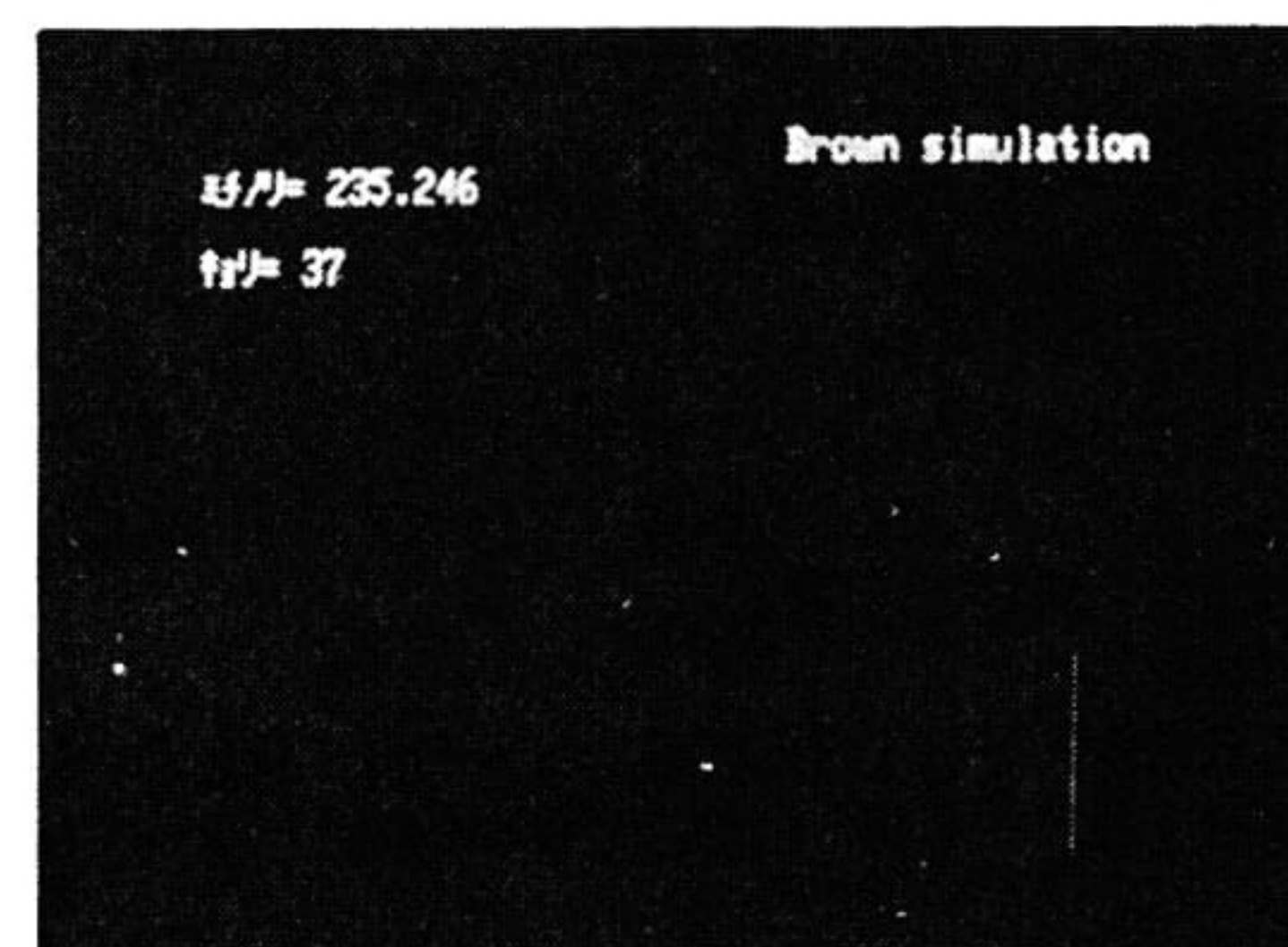
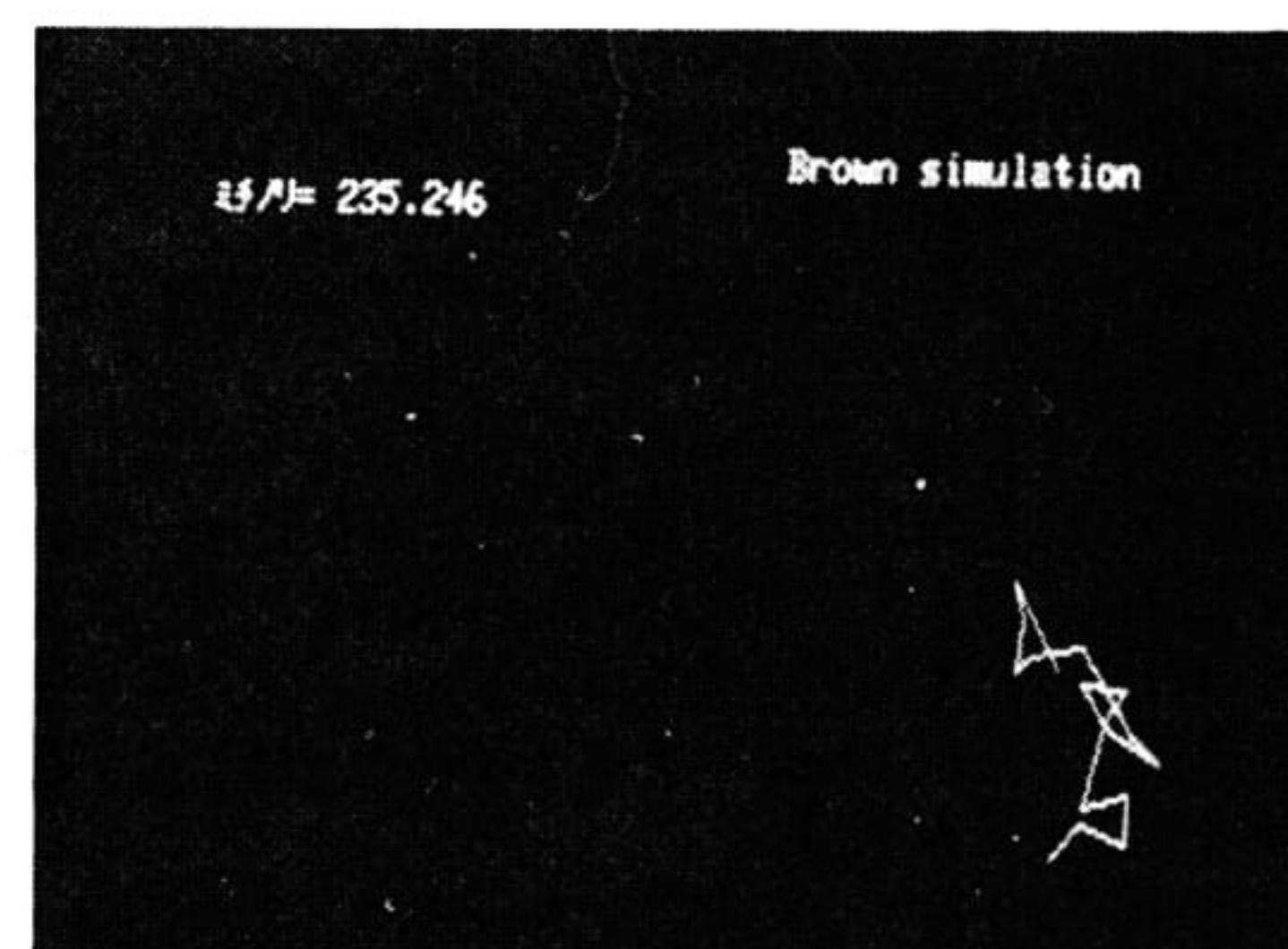
パラメータが0と1のときは，LPの値をワールド座標上での値として，それぞれX座標とY座標を与えます．LPを設定する方のPOINT命令は，絶対座標で指定するときのパラメータにワールド座標を用いていました．したがって，

POINT (320, 250) : PRINT POINT (0), POINT (1)

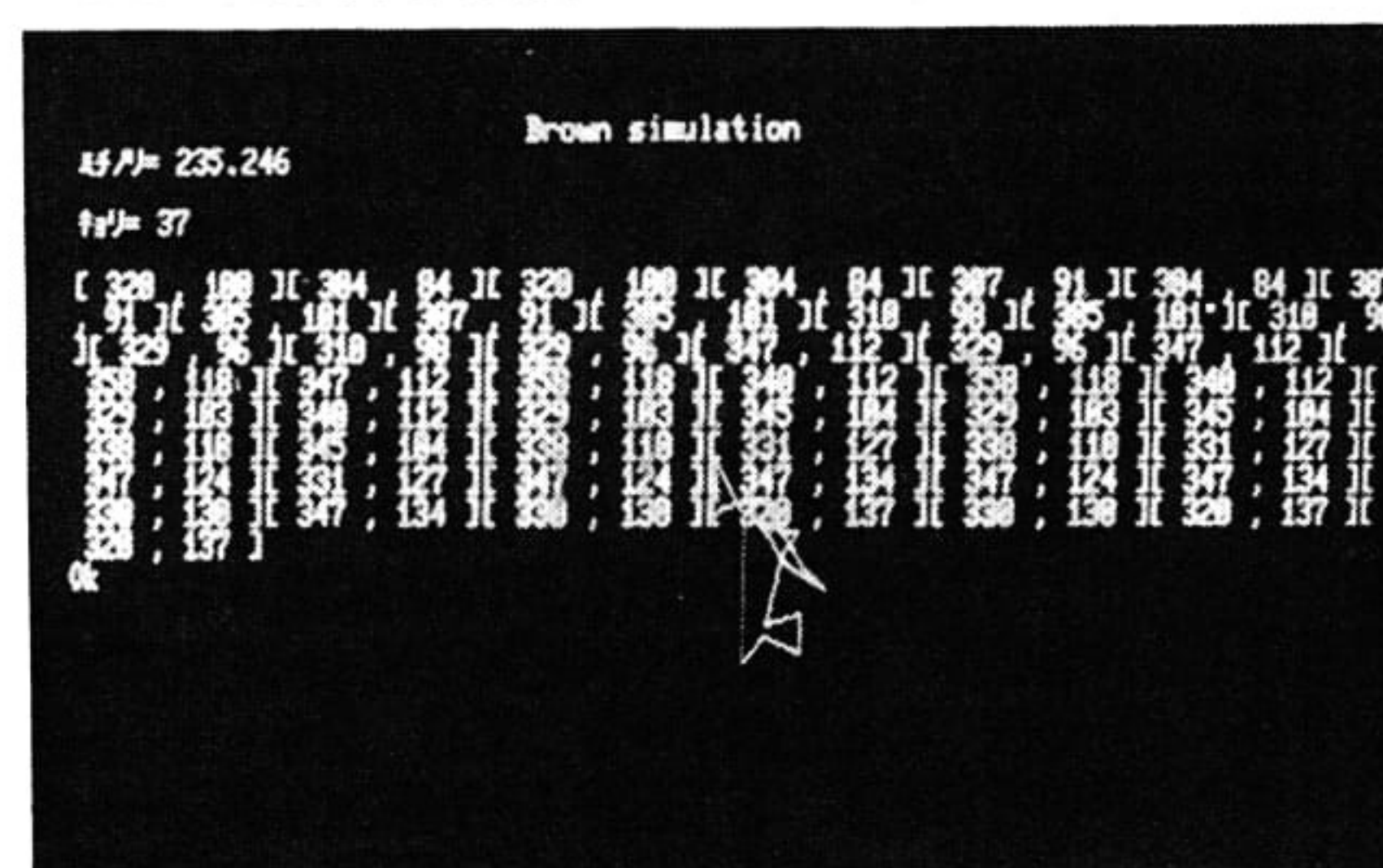
とすれば，当然，画面には，320と250という値が表示されます．相対座標指定を多用した後などに，LPの値を知る必要がある時などにも便利だと思います．

例えばプログラム30のようなブラウン運動のシミュレーションを考えてみましょう．ブラウン運動について，詳しい話は専門書に任せることにして，ここでは，単に平面上での粒子の不規則な運動と思っていただければ結構です．ある程度実行させたら，何かキーを押して下さい．粒子（画面上の緑色の点）の動き回った跡が青い線と，その道程が表示されたと思います．次にもう一度，何かキーを押すと，粒子の始点と終点を結ぶ赤い線と，その距離が表示されたと思います．これで実際に粒子が進んだ距離が求められた訳です．この距離を求めるのに，現在のLPの値が必要になってくる訳です．最後にもう一度，何かキーを押して下さい．

パラメータが2と3のときは，LPの値をスクリーン座標上での値として，それぞれX座標とY座標を与えます．こちらの方は，WINDOW命令やVIEW命令がややこしい設定の仕方になっていると，人間の手計算では，多少厄介です．

```
100 '
110 '   Program 30
120 '     Brown simulation
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  RANDOM=VAL(RIGHT$(TIME$,2))
170  RANDOMIZE(RANDOM)
180  LOCATE 27,0,0 : PRINT "Brown simulation"
190  STARTX=320 : STARTY=100
200  PSET(STARTX,STARTY),4
210  GOSUB *PALETTEINIT
220  COLOR=(1,0) : COLOR=(2,0)
230 '
240 *LOOP
250  NEXTX=RND(1)*40-20
260  NEXTY=RND(1)*40-20
270  DISTANCE=DISTANCE+SQR(NEXTX^2+NEXTY^2)
280  PSET STEP(NEXTX,NEXTY),4
290  FOR DELAY=0 TO 20 : NEXT
300  PRESET STEP(-NEXTX,-NEXTY)
310  LINE -STEP(NEXTX,NEXTY),1
320  IF INKEY$="" THEN *LOOP
330 '
340  COLOR=(1,1)
350  PRINT "ﾐﾁﾉﾘ=";DISTANCE
360  PRINT
370  IF INKEY$="" THEN 370
380 '
390  COLOR=(1,0) : COLOR=(2,2)
400  ENDX=POINT(0) : ENDY=POINT(1)
410  DISTANCE=SQR((STARTX-ENDX)^2+(STARTY-ENDY)^2)
420  LINE(STARTX,STARTY)-(ENDX,ENDY),2
430  PRINT "ｷｮﾘ=";DISTANCE
440  PRINT
450  IF INKEY$="" THEN 450
```

```
460 '
470  GOSUB *PALETTEINIT
480  LOCATE ,,1
490 END
500 '
510 *PALETTEINIT
520  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
530 RETURN
```

プログラム31に，サブルーチン・パッケージとして＊GTRACE というのを作ってみました．BASIC の中で，TRON という命令がありますが，これと同じ様なプログラムです．

```
100 '
110 '   Program 31
120 '     Brown simulation
130 '
140 '
145 DIM RESULT(1,50) : MODE$="W" : RESULT(0,0)=1
150  SCREEN 0,0 : CLS 3
160  RANDOM=VAL(RIGHT$(TIME$,2))
170  RANDOMIZE(RANDOM)
180  LOCATE 27,0,0 : PRINT "Brown simulation"
190  STARTX=320 : STARTY=100
200  PSET(STARTX,STARTY),4 : GOSUB *GTRACE
210  GOSUB *PALETTEINIT
220  COLOR=(1,0) : COLOR=(2,0)
230 '
240 *LOOP
250  NEXTX=RND(1)*40-20
260  NEXTY=RND(1)*40-20
270  DISTANCE=DISTANCE+SQR(NEXTX^2+NEXTY^2)
280  PSET STEP(NEXTX,NEXTY),4 : GOSUB *GTRACE
290  FOR DELAY=0 TO 20 : NEXT
300  PRESET STEP(-NEXTX,-NEXTY) : GOSUB *GTRACE
310  LINE -STEP(NEXTX,NEXTY),1 : GOSUB *GTRACE
320  IF INKEY$="" THEN *LOOP
330 '
340  COLOR=(1,1)
350  PRINT "ﾐﾁﾉﾘ=";DISTANCE
360  PRINT
370  IF INKEY$="" THEN 370
380 '
390  COLOR=(1,0) : COLOR=(2,2)
400  ENDX=POINT(0) : ENDY=POINT(1)
410  DISTANCE=SQR((STARTX-ENDX)^2+(STARTY-ENDY)^2)
420  LINE(STARTX,STARTY)-(ENDX,ENDY),2 : GOSUB *GTRACE
430  PRINT "ｷｮﾘ=";DISTANCE
440  PRINT
450  IF INKEY$="" THEN 450
460 '
470  GOSUB *PALETTEINIT
480  LOCATE ,,1
485  GOSUB *TEND
490 END
500 '
510 *PALETTEINIT
520  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
530 RETURN
540 '
1000 *GTRACE : IF RESULT(0,0)>50 THEN RETURN
1010  IF MODE$="W" THEN RESULT(0,RESULT(0,0))=POINT(0)
                    :RESULT(1,RESULT(0,0))=POINT(1)
1020  IF MODE$="S" THEN RESULT(0,RESULT(0,0))=POINT(0)
                    :RESULT(1,RESULT(0,0))=POINT(1)
1030  RESULT(0,0)=RESULT(0,0)+1
1040 RETURN
1050 '
1060 *TEND
1070  IF MODE$="W" THEN S$="[" : E$="]" ELSE S$="(" : E$=")"
1080  FOR I=1 TO RESULT(0,0)-1
1090    PRINT S$;RESULT(0,I);",";RESULT(1,I);E$;
1100  NEXT
1110 RETURN
```

しかし，＊GTRACEは，グラフィック命令のLPをトレースするものです．使い方を述べますと，まず，グラフィック命令を用いた何かのプログラムの後にでも，この＊GTRACEをつなげて下さい．そして，＊GTRACEの実用に欠く点ですが，つなげたプログラム中のトレースしたいグラフィック命令の後に，GOSUB　＊GTRACEと付け足して，プログラムの最初にも，DIM RESULT（1，50）：MODE$＝〝機能〟　と付け足してください．機能の所は，〝W〟とするとワールド座標上，〝S〟とするスクリーン座標上におけるLPの値を，RESULTという配列に格納していきますが，格納できるのは50個までです．そしてもう一つ，プログラムの最後に，GOSUB ＊TEND　と付け足して下さい．トレースの結果を画面に表示します．この時出力している座標は，ワールド座標だと，［　］で括られ，スクリーン座標だと（　）で括られます．実用にはほど遠い感じですが，実験のつもりで試してみるのも一興でしょう．プログラム31では，プログラム30につけた例です．

POINT関数は，もう一つの機能を持っています．分かり易く考えるならば，今までとは異なる機能を持ったPOINT関数がもう一つある，と言い換えてもいいでしょう．この関数は，

POINT（X，Y）

と書きます．パラメータ（X，Y）は，スクリーン座標で指定してやります．関数の与える値は，（X，Y）で指定した，スクリーン座標上におけるドットの色のパレット番号です．もし，（X，Y）がビューポートの外側を指定していた時は，－1を値として返してきます．あくまでスクリーン座標で指定するものですから，ワールド座標とスクリーン座標が一致していないときは，ワールド座標上におけるドットの色のパレット番号を知ることができません．知るためには，スクリーン座標とワールド座標を変換する必要があります．このにない手となる関数が，MAP関数です．

プログラム32は，MAP関数のワールド座標からスクリーン座標への変換と同じことをしています．MAP関数の書式は，

MAP（C，F）

となっていて，プログラム30における変数C，Fも同じ意味を持っています．（表15）は，パラメータ別に，MAP関数の機能を示しています．

```
100 '
110 '  Program 32
120 '    imitative MAP function
130 '    C: --> co-ordinate value
140 '    F: --> map parameter (0-1)
150 '
160 *IMTATIVEMAP
170  STACKX=POINT(0)  : STACKY=POINT(1)
180  POINT(C,C)  : MAPVAL=POINT(F+2)
190  POINT(STACKX,STACKY)
200 RETURN
```

F	Cの値の解釈のされ方	関数が返す値
0	ワールド座標上のX座標	スクリーン座標上のX座標
1	ワールド座標上のY座標	スクリーン座標上のY座標
2	スクリーン座標上のX座標	ワールド座標上のX座標
3	スクリーン座標上のY座標	ワールド座標上のY座標

表15 MAP(C, F)のパラメータと変換機能の関係

以上に説明した，POINT 関数と MAP 関数を用いて，簡単なプログラムを組んでみましょう．点を打つ前に必ず，*COLORCHECK を実行しています．*COLORCHECK は，MAP 関数を用いてワールド座標（PX, PY）をスクリーン座標（SX, SY）に変換し，そのスクリーン座標で示された点の色を，POINT 関数によって求めています．そして，もし既に点が打たれていたら，その点の色を変えます．この処理をしながら線を描くので，一風変わった模様ができるわけです．

カラーページ参照

```
100 '
110 '   Program 33
120 '     MAP & POINT function demonstration
130 '
140 '
150  SCREEN 0,1 : CLS 3
160  WINDOW(0,-100)-(200,0)
170  VIEW(220,50)-(441,150),,7
180 '
190  X2=200
200  FOR Y2=0 TO -100 STEP -1
210    GOSUB *LINEDRAW
220  NEXT
230  FOR X2=200 TO 0 STEP -1
240    GOSUB *LINEDRAW
250  NEXT
260 END
270 '
280 *LINEDRAW
290  C=1
300  IF X1>X2 THEN SWAP X1,X2 : SWAP Y1,Y2
310  IF X1=X2 THEN *EQUALX
320  DX=X2-X1 : DY=Y2-Y1
330  FOR PX=X1 TO X2
340    PY=(PX-X1)*DY/DX+Y1
350    GOSUB *COLORCHECK
360    PSET(PX,PY),C : C=1
370  NEXT
380 RETURN
390 '
400 *EQUALX
410  IF Y1>Y2 THEN SWAP Y1,Y2
420  FOR PY=Y1 TO Y2
430    GOSUB *COLORCHECK
440    PSET(PX,PY),C : C=1
450  NEXT
460 RETURN
470 '
480 *COLORCHECK
490  SX=MAP(PX,0) : SY=MAP(PY,1)
500  IF POINT(SX,SY)>0 THEN C=2
510 RETURN
```

この程度のことでは，MAP 関数の有用性を理解するのは困難だと思います．その真価がはっきりするのは，スクリーン座標とワールド座標が，組み合わされた時でしょう．したがって，スクリーン座標によって座標指定を行う，次の節で紹介する GET＠と PUT＠と共に用いたときに，有用性が分かると思います．

4.3.9 GET＠，PUT＠―グラフィックパターン，漢字を操る―

VIEW 命令で，星を描く位置を決めるというのは，良い考えでした．しかし，全く同じ大きさ，同じ形のものを数多く描くとき，星などの場合は，描く度に無駄な時間を計算に使っています．そこで無駄な計算を1回で済ませるのが，GET＠とPUT＠を組み合わせた使用法です．例として，アメリカの国旗を描いてみましょう．

星を描くサブルーチンは，＊STARDRAW です．180行で，1回，＊STARDRAW を呼び出して大きな星を，画面のほぼいっぱいに描きます．この星を塗りつぶす様に，220行から270行で，赤と白のストライプを描き，国旗の下地を作成します．290行で国旗の青い部分を描けば，下準備は完了です．次はいよいよ 50個の星を描きます．

まず，小さい星を描くために，340行で VIEW 命令を用いて，星の縮小した図を描ける様にして，＊STARDRAW を呼び出します．星を描くための計算をすることをこれ以後，省くためには，この図をどこかにしまっておいて，描こうとする時，しまってある所から引き出してきて．任意の位置に写せばいいのです．

図をしまう命令がGET＠命令で，しまう場所は数値型の配列変数の中です．390行で，GET＠を使っています．ここで座標の指定は，スクリーン座標です．したがって，380行の，SCREEN 0，0というのは，スクリーン座標の初期化（スクリーン座標を，グラフィック画面いっぱいの大きさにすること）を行って，390行で星を描くために，VIEW 命令で切りとった部分をそのまま配列変数 STAR%にしまう準備のようなものです．

ところで，この星をしまうために，どの程度の大きさの配列変数が必要でしょうか？　その大きさを計算しているのは，370，380行です．まず必要なバイト数を，370行で変数Pに求めており，そのために次のような公式を使います．

必要なバイト数＝（(横のドット数＋7）￥8）＊縦のドット数＊M＋4

この公式におけるMは，白黒モードのとき1，カラーモードのとき3です．プログラム34では，カラーモードですから，M＝3で計算します．横と縦のそれぞれのドット数は，390行の GET＠を例とすると，

横のドット数＝30－5＋1＝26
縦のドット数＝12－2＋1＝11

と計算して求めます．

配列変数は，配列の中の１つの変数が１バイトではないので，必要なバイト数が分かっても，添字の値はまだ決まりません．添字の値を求めるには，次の公式を利用します．

添字の値＝必要なバイト数¥N＋1

この時のNの値が，配列変数の型によって左右されるもので，（表14）の通りです．配列変数は，DIM 文で宣言する必要がありますから，380行でその手続きをしています．配列変数名の終わりに％が付いていますから，この場合の配列は整数型配列です．したがって，N＝２を用いて計算します．（但し，添字の値は，一次元配列で添字の底が０のときの最小値です）．

GET＠については，以上のことを知っていれば，たいていのことは不自由しないでしょうが，もう一つパラメータがあるので説明しておきます．次の書式を見て下さい．

GET＠(X１，Y１)−（X２，Y２)，配列変数名，要素

最後に，要素というパラメータがあります．この値は，配列のどの要素からグラフィックのパターンをしまうかを指定するもので，当然，省略した時は最初からしまいます．これは，１つの配列に，複数のグラフィック・パターンをしまうとき使用します．

配列の型	N
整 数 型	2
単精度型	4
倍精度型	8

表16 配列の型とバイト数

条 件	機 能
PSET	そのまま表示する．
PRESET	白黒モードの場合は，リバースして表示する． カラーモードの場合は，各ドットのパレット番号を7−(ドットのパレット番号）として表示する．
OR	ドットごとに，画面上のパターンとORをとり，表示する．
AND	ドットごとに，画面上のパターンとANDをとり，表示する．
XOR	ドットごとに，画面上のパターンとXORをとり，表示する．

この条件は画面のモードによって異なった演算を行う．
白黒モード──→ドットの有無(I or 0)で演算．
カラーモード──→パレット番号で演算．

表17 PUT＠における条件のパラメータと働き

さて，GET＠で配列変数にしまったグラフィック・パターンを画面の任意の場所に表示しましょう．そのために用いる命令が，PUT＠です．PUT＠命令も，GET＠命令と同様に，座標指定はスクリーン座標によって行います．一般の書式は，

PUT＠(X１，Y１)，配列変数名 （要素ナンバ)，条件，フォア・グラウンド・カラー
，バック・グラウンド・カラー

となっています．配列変数名は，表示したグラフィック・パターンのしまってあるものを指定し

ます．要素ナンバは，配列内のどこから表示するかをGET＠で用いたものと同じもので指定します．もちろん，省略すれば配列の最初から取り出して表示します．次の条件というパラメータは，（表15）にあるうちの１つを選ぶことができます（省略した場合は，XORとなります）．使用の実例は，プログラム34の430行，490行を見て下さい．この２行は，『スクリーン座標（K，L）の示す点を左上とした長方形の中に，配列STAR%にしまわれているグラフィックパターンを，そのまま表示する』という意味です．410行から510行まで２つのFOR～NEXTのループで，50個の星を描き終ります．

カラーページ参照

```
100 '
110 '  Program 34
120 '    the Stars and Stripes
130 '
140 '
150  SCREEN 0,3 : CLS 3
160  SCREEN ,0 : WINDOW(-300,-300)-(300,300)
170  VIEW(90,0)-(549,194)
180  GOSUB *STARDRAW
190 '
200 '  stripes drawing
210 '
220  WINDOW(0,0)-(639,194) : VIEW(0,0)-(639,194)
230  POINT(639,-1)
240  FOR I=0 TO 6
250    LINE STEP(-639,1)-STEP(639,14),2,BF
260    LINE STEP(-639,1)-STEP(639,14),7,BF
270  NEXT
280 '
290  LINE(0,0)-(320,110),1,BF
300 '
310 '  50 stars drawing
320 '
330  WINDOW(-300,-300)-(300,300)
340  VIEW(5,2)-(30,12)
350  GOSUB *STARDRAW
360 '
370  P=((26+7)¥8)*11*3+4
380  DIM STAR%(P¥2+1) : SCREEN 0,0
390  GET@(5,2)-(30,12),STAR%
400 '
410  FOR L=2 TO 98 STEP 24
420    FOR K=5 TO 280 STEP 55
430      PUT@(K,L),STAR%,PSET
440    NEXT
450  NEXT
460 '
470  FOR L=14 TO 86 STEP 24
480    FOR K=32 TO 252 STEP 55
490      PUT@(K,L),STAR%,PSET
500    NEXT
510  NEXT
520 END
530 '
540 '  subroutine (star pattern draw)
550 '
560 *STARDRAW
570  K=2*3.1415/5
580  SX=COS(K) : SY=SIN(K)
590  X=0 : Y=-290 : POINT(X,Y)
600  FOR I=1 TO 5
610    FOR J=1 TO 2
620      P=X : X=SX*X-SY*Y
630      Y=SY*P+SX*Y
640    NEXT
650    LINE -(X,Y),7
660  NEXT
670 '
```

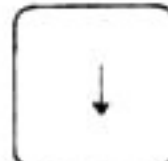
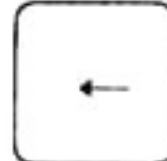
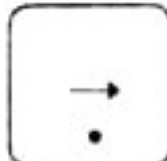
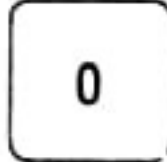
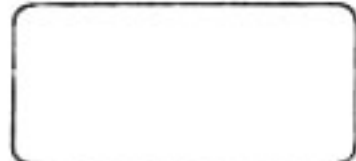
```
680   PAINT(0,0),7,7
690   X=0 : Y=-140
700   FOR I=1 TO 5
710      P=X : X=SX*X-SY*Y
720      Y=SY*P+SX*Y
730      PAINT(X,Y),7,7
740   NEXT
750  RETURN
```

この程度のことをするのに，1つの配列の中に複数のパターンがしまえる必要はないのでは？と考える人がいると思います．では，有効な使い方を示して，その必要性について考えてみましょう．

プログラム35は，デジタル時計を作るものです．PC-8801は，システム変数として TIME$(現在の時刻を示す)を持っています．TIME$ の示す時刻を単に，キャラクタで表示するのでは味気ないので，オリジナルの数字を作ってこれを数値と対応させて表示させます．この対応をうまく処理してくれるのが，PUT@命令の要素ナンバーなのです．プログラムの＊LOOP 以降の FOR ループの中を見て下さい．

```
100 '
110 '   Program 35
120 '     put@ demonstration
130 '
140 '
150  SCREEN 0,0 : CLS 3
160  DIM DIGIT%(4330)
170 '
180  RESTORE *SEGARRAY
190  FOR I=0 TO 9
200    READ E(I),F$(I)
210  NEXT
220 '
230  FOR L=0 TO 9
240    FOR I=1 TO E(L)
250      S=VAL(MID$(F$(L),I,1))
260      ON S+1 GOSUB *SSEG0,*SSEG1,*SSEG2,*SSEG3,
                      *SSEG4,*SSEG5,*SSEG6
270    NEXT
280    P=L*433
290    GET@(0,0)-(49,40),DIGIT%(P)
300    LINE(0,0)-(49,40),0,BF
310  NEXT
320 '
330  GOSUB *DECORATE
340 '
350 *LOOP
360  O$=T$:T$=TIME$
370  PX=100
380  FOR I=1 TO 8
390    IF MID$(T$,I,1)=":" THEN 440
400    IF MID$(T$,I,1)=MID$(O$,I,1) THEN 440
410    D%=VAL(MID$(T$,I,1))
420    P=D%*433
430    PUT@(PX,80),DIGIT%(P),PSET
440    PX=PX+50
450  NEXT
460 GOTO *LOOP
470 '
480 *SEG0
490  DATA 9,2,35,0,-6,4,-22,0,19,5
500 *SEG1
510  DATA 7,4,-2,15,6,-3,2,-7,9,7
520 *SEG2
530  DATA 46,4,-2,15,-6,-3,2,-7,45,7
```

```
540 *SEG3
550  DATA 5,21,-2,15,6,-3,2,-7,7,28
560 *SEG4
570  DATA 44,21,-2,15,-6,-3,2,-7,42,28
580 *SEG5
590  DATA 5,38,35,0,-6,-4,-22,0,19,35
600 *SEG6
610  DATA 7,20,6,-2,22,0,6,2,-6,2,-22,0,24,20
620 '
630 *SSEG0
640  RESTORE *SEG0
650  GOSUB *DRAW
660 RETURN
670 '
680 *SSEG1
690  RESTORE *SEG1
700  GOSUB *DRAW
710 RETURN
720 '
730 *SSEG2
740  RESTORE *SEG2
750  GOSUB *DRAW
760 RETURN
770 '
780 *SSEG3
790  RESTORE *SEG3
800  GOSUB *DRAW
810 RETURN
820 '
830 *SSEG4
840  RESTORE *SEG4
850  GOSUB *DRAW
860 RETURN
870 '
880 *SSEG5
890  RESTORE *SEG5
900  GOSUB *DRAW
910 RETURN
920 '
930 *SSEG6
940  RESTORE *SEG6
950  GOSUB *DRAWS
960 RETURN
970 '
1000 *DRAW
1010  READ X1,Y1 : POINT(X1,Y1)
1020  FOR B=1 TO 3
1030    READ X2,Y2
1040    LINE -STEP(X2,Y2),4
1050  NEXT
1060  LINE -(X1,Y1),4
1070  READ X2,Y2
1080  PAINT(X2,Y2),4,4
1090 RETURN
1100 '
1110 *DRAWS
1120  READ X1,Y1 : POINT(X1,Y1)
1130  FOR B=1 TO 5
1140    READ X2,Y2
1150    LINE -STEP(X2,Y2),4
1160  NEXT
1170  LINE -(X1,Y1),4
1180  READ X2,Y2
1190  PAINT(X2,Y2),4,4
1200 RETURN
1210 '
1220 *DECORATE
1230  LINE(60,60)-(540,140),1,BF
1240  LINE(80,70)-(520,130),0,BF
1250  LINE(20,40)-(580,160),1,B
1260  PAINT(40,50),CHR$(0)+CHR$(&HF0)+STRING$(2,CHR$(0))+CHR$(&HF)
      +CHR$(0),1
1270  FOR CENTERX=225 TO 375 STEP 150
1280    FOR CENTERY=90 TO 110 STEP 20
1290      CIRCLE(CENTERX,CENTERY),5,4
```

```
1300        PAINT(CENTERX,CENTERY),4,4
1310      NEXT
1320   NEXT
1330 RETURN
1340 '
1350 *SEGARRAY
1360   DATA 6,"013542",2,"24",5,"02635",5,"02645",4,"1624"
1370   DATA 5,"01645",6,"013546",4,"1024",7,"0164532",6,"016245"
```

まず，D%にT$の中の1文字を数値に変換して代入しています．当然，この値は0から9までの値です．この数値を倍すれば，*LOOP以前で，GET@によって作成した数字のパターンが，しまってあるそれぞれの場所の先頭の要素ナンバーと一致します．したがって，PUT@によってこれを表示すれば，数値と同じグラフィック・パターンの表示ができる訳です．もし，数字のグラフィック・パターンを10個異なった配列にしまって，これを数値と対応させるとしたら，おそらくプログラム35と比べて，かなり複雑な処理になるでしょう．

このプログラムでは，用いていないパラメータである，フォアグラウンド・カラーと，バック・グラウンド・カラーは，両方とも指定するか，指定しないかのどちらかに限られます．したがって，一方のみ省略することはできません．この2つの色指定のパラメータは，パレット番号で指定するもので，白黒モードで読み込んだものに色を付けます．フォアグラウンド・カラーは，白黒モードのときの白の部分を，バック・グラウンド・カラーは，黒の部分を，色指定するものです．

次に，PUT@命令を用いて，画面上にカーソルを出してみましょう．カーソルの形は，十字形をしています．ただ単に，カーソルを出しても，面白くないですから，このカーソルを画面上の任意の場所へ移動させることができ，さらに，そのカーソルの着目している点に関して，今まで話をしたグラフィック命令が，実行されるようなプログラムにします．このときのカーソルを，特にGカーソル（Graphic Cursor）と呼ぶことにします（プログラム36）．

さて，カーソルを自由に移動させて，図形を描くために，キーを押して指示を与える訳ですが，次に，キーの意味を説明しておきましょう．

[↑] [↓] [←] [→]：Gカーソルを矢印の方向に移動させます．処理している行番号

[0]〜[9]：カーソルの移動量を変えます．

[　]：フォア·グラウンド·カラーのパレット番号が1だけ増えます．但し，既にパレット番号が7であった時は，0になります．*COLORRで処理を行っており，変数C%がフォア·グラウンド·カラーを保持しています．

[I]：Informationの意味です．*INFOで処理をしており，LPの座標，カーソルの着目している座標，フォア·グラウンド·カラーの値を知ることができます．

プログラム中の変数には，次のものが重要です．

X，Y　：現在，カーソルの着目している座標．

LPX, LPY：LPの座標．但し，これはN_{88}が持っているLPとは異ったものですが，機能的には全く同じものです．

C％　　：フォア･グラウンド・カラー．

では，残りのキーの各処理をしているサブルーチン名を次に挙げておきます．各ルーチンを見れば，そのキーによって実行されることは明らかです．

[Sト] ＊PSETR

[Rス] ＊PRESETR

[Lリ] ＊LINER

[Bコ] ＊LINEB

[Fハ] ＊LINEBF

[Cソ] ＊CIRCLER 但し，半径は LPと現在着目している座標との距離になります．

[Pセ] ＊PAINTR 3通りの方法に分かれます． それぞれ，聞かれていることに，答えていって下さい．但し，Middle color は，数値（0～101）を入力すれば，中間色を勝手に生成しますが，Tiling は，y方向のドット指定すると，その分，タイル･パターンを聞いてきますので，これには16進数で答えて下さい．

[Gキ] ＊GETR 絵や図形を一種類だけ覚えることができます． 大きさの目安としては，50ドット×50ドット以内におさまる程度です．

[Mモ] ＊PUTR 覚えている絵や図形（[Gキ]で覚えたもの）を画面に再生します．

[Eイ] ＊CLSR 画面（グラフィック画面）を消します．

[Aチ] ＊LPSET Last Reference Point を変更します．

[⏎]で作図をやめます．作成した画面は，プリンタにコピーすることも（但し，白黒になりますが……）できますし，さらに書き加えることもできます．以上で，プログラム36の説明としますが，もっと使い易く，あるいは，機能も多くなど，改良の余地はいくらでもありますから，自分だけのテレビ黒板を作ってみたらいかがでしょうか．

カラーページ参照

```
100 '
110 '  Program 36
120 '
130 '
140  ON ERROR GOTO *TRAP
150  WIDTH 80,25 : CONSOLE 20,5,0,1
160  SCREEN 0,1 : COLOR 7,0,0,7
170 '
180  DEFINT A-Z
190  DIM CU%(75),PICTURE%(530),COMMAND%(13),RGB$(2)
200  GOSUB *COLORINIT
210 '
220  CLS 3
230  SCREEN 0,2
240  VIEW(0,0)-(55,25)
250  LINE(0,10)-(50,10),7
260  LINE(25,0)-(25,20),7
270  GET@(17,6)-(32,14),CU%
280  SCREEN 0,0 : CLS 2
```

```
290 '
300  RESTORE *STEPDATA
310  FOR I=1 TO 9
320    READ DX(I)
330  NEXT
340  FOR I=1 TO 9
350    READ DY(I)
360  NEXT
370  FOR I=1 TO 13
380    READ COMMAND%(I)
390  NEXT
400 '
410 *STEPDATA
420  DATA 1,2,4,8,16,32,64,128,256
430  DATA 1,2,3,4,8,16,32,64,128
440  DATA &H20,&H73,&H72,&H6c,&H62,&H66,&H63
450  DATA &H70,&H6d,&H67,&H65,&H61,&H69
460 '
470  INC0=1 : INC1=1
480 '
490 *DRAWING
500  LOCATE 0,20 : PRINT "Welcome !"
510  X=320 : Y=100
520  GOSUB *LPSET
530  C=7
540  PUT@(X-8,Y-4),CU%,XOR
550  GOSUB *GETKEY
560 '
570 *LOOP1 : IF CH=13 THEN *EXIT1
580     IF (CH>&H30) AND (CH<=&H39) THEN
          ID=CH-&H30 : INC0=DX(ID) : INC1=DY(ID) : GOTO *GETNEXT
590     PUT@(X-8,Y-4),CU%,XOR
600     IF (CH<28) OR (CH>=32) THEN *OTHER
610     IF CH=29 THEN X=X-INC0 : IF X-8<0 THEN X=627
620     IF CH=28 THEN X=X+INC0 : IF X-8>619 THEN X=8
630     IF CH=30 THEN Y=Y-INC1 : IF Y-4<0 THEN Y=193
640     IF CH=31 THEN Y=Y+INC1 : IF Y-4>189 THEN Y=4
650     PUT@(X-8,Y-4),CU%,XOR
660     GOTO *GETNEXT
670     F=1
680    *OTHER : IF COMMAND%(F)=CH THEN *EXIT2
690     F=F+1 : IF F>13 THEN F=-1 ELSE *OTHER
700    *EXIT2 : IF F=-1 THEN *SKIP
710      ON F GOSUB *COLORR,*PSETR,*PRESETR,*LINER,*LINEB,*LINEBF,
                   *CIRCLER,*PAINTR,*PUTR,*GETR,*CLSR,*LPSET,*INFO
720      *SKIP
730       PUT@(X-8,Y-4),CU%,XOR
740      '
750    *GETNEXT GOSUB *GETKEY
760  GOTO *LOOP1
770 *EXIT1 : PUT@(X-8,Y-4),CU%,XOR
780  PRINT "Copy to printer ";:C$=INPUT$(1)
790  IF C$<>"y" THEN *MORE
800  PRINT "Prepare your printer !"
810  PRINT "press any key."
820  IF INKEY$="" THEN 820
830  COPY 2
840 '
850 *MORE
860  PRINT : PRINT "Retry ";:C$=INPUT$(1)
870  IF C$="y" THEN *DRAWING
880  CLS : PRINT "See you again !"
890  CONSOLE 0,25
900 END
910 '
920 *GETKEY
930  CH=ASC(INPUT$(1))
940 RETURN
950 '
960 *COLORR
970  C=(C+1) MOD 8
980  COLOR ,,,C : PRINT "Fore ground color = ";C
990 RETURN
1000 '
```

```
1010 *PSETR
1020  PSET(X,Y)
1030  GOSUB *LPSET
1040 RETURN
1050 '
1060 *PRESETR
1070  PRESET(X,Y)
1080  GOSUB *LPSET
1090 RETURN
1100 '
1110 *LINER
1120  LINE(LPX,LPY)-(X,Y)
1130  GOSUB *LPSET
1140 RETURN
1150 '
1160 *LINEB
1170  LINE(LPX,LPY)-(X,Y),,B
1180  GOSUB *LPSET
1190 RETURN
1200 '
1210 *LINEBF
1220  LINE(LPX,LPY)-(X,Y),,BF
1230  GOSUB *LPSET
1240 RETURN
1250 '
1260 *CIRCLER
1270  RADIUS=SQR((X-LPX)^2+(Y-LPY)^2)
1280  CIRCLE(X,Y),RADIUS
1290  GOSUB *LPSET
1300 RETURN
1310 '
1320 *PAINTR
1330  PRINT
1340  PRINT "1) Normal color"
1350  PRINT "2) middle color"
1360  PRINT "3) tiling" : PRINT
1370  PRINT "Your choice - ";:A=ASC(INPUT$(1))
1380  IF A=13 THEN RETURN
1390  IF A=&H31 THEN *NORMAL
1400  IF A=&H32 THEN *MIDDLE
1410  IF A=&H33 THEN *TILING
1420  GOTO *PAINTR
1430 '
1440 *NORMAL
1450  CLS : PRINT "Normal color"
1460  INPUT "Paint color,border color(0-7)";PC,BC
1470  IF PC<0 AND PC>7 THEN RETURN
1480  IF BC<0 AND BC>7 THEN RETURN
1490  PAINT(X,Y),PC,BC
1500  GOSUB *LPSET
1510 RETURN
1520 '
1530 *MIDDLE
1540  CLS
1550  INPUT "Middle color(0-101)";MC
1560  INPUT "Border color(0-7)";BC
1570  IF MC<0 OR MC>101 THEN RETURN
1580  IF BC<0 OR BC>7 THEN RETURN
1590  BLUE=COL(MC,1)
1600  RED=COL(MC,2)
1610  GREEN=COL(MC,3)
1620  TILE$=""
1630  FOR I=0 TO 1
1640    TILE$=TILE$+CHR$(TL(BLUE,I))+CHR$(TL(RED,I))+CHR$(TL(GREEN,I))
1650  NEXT
1660  BACK$=STRING$(3,CHR$(0))
1670  IF LEFT$(TILE$,3)=BACK$ THEN BACK$=STRING$(3,CHR$(1))
1680  PAINT(X,Y),TILE$,BC,BACK$
1690  GOSUB *LPSET
1700 RETURN
1710 '
1720 *TILING
1730  CLS : PRINT "Tiling"
1740  INPUT "Max y dot ";YDOT
1750  INPUT "Border color(0-7)";BC
```

```
1760  IF BC<0 AND BC>7 THEN RETURN
1770  TILE$=""
1780  FOR I=1 TO YDOT
1790    INPUT "Blue pattern &H",RGB$(0)
1800    INPUT "Red pattern &H",RGB$(1)
1810    INPUT "Green pattern &H",RGB$(2)
1820    FOR J=0 TO 2
1830      TILE$=TILE$+CHR$(VAL("&H"+RGB$(J)))
1840    NEXT
1850  NEXT
1860  PAINT(X,Y),TILE$,BC,STRING$(3,CHR$(0))
1870  GOSUB *LPSET
1880 RETURN
1890 '
1900 *PUTR
1910  PUT@(X,Y),PICTURE%,XOR
1920  GOSUB *LPSET : PRINT "put - ok!"
1930 RETURN
1940 '
1950 *GETR
1960  IF ABS(X-LPX)>50 THEN RETURN
1970  IF ABS(Y-LPY)>50 THEN RETURN
1980  GET@(LPX,LPY)-(X,Y),PICTURE%
1990  GOSUB *LPSET : PRINT "get - ok!"
2000 RETURN
2010 '
2020 *CLSR
2030  CLS 2
2040 RETURN
2050 '
2060 *LPSET
2070  LPX=X : LPY=Y
2080 RETURN
2090 '
2100 *INFO
2110  PRINT "Last reference point (";LPX;",";LPY;")"
2120  PRINT "Cursor point (";X;",";Y;")"
2130  PRINT "Fore ground color (palette number = ";C;" )"
2140 RETURN
2150 '
2160 *COLORINIT
2170  DIM TL(4,1),COL(102,3)
2180  RESTORE *TILEDATA
2190  FOR I=0 TO 4
2200    READ TL(I,0),TL(I,1)
2210  NEXT
2220  NUM=1 : ST=0
2230  FOR ED=1 TO 3
2240    GOSUB *COLINIT
2250  NEXT
2260  FOR ST=0 TO 3
2270    GOSUB *COLINIT
2280  NEXT
2290  FOR BLUE=0 TO 4
2300    RED=BLUE : GREEN=RED
2310    GOSUB *CINTSUB
2320  NEXT
2330 RETURN
2340 '
2350 *COLINIT
2360  BLUE=ST : RED=ED
2370  FOR GREEN=ST TO ED-1
2380    GOSUB *CINTSUB
2390  NEXT
2400  FOR RED=ED TO ST+1 STEP -1
2410    GOSUB *CINTSUB
2420  NEXT
2430  FOR BLUE=ST TO ED-1
2440    GOSUB *CINTSUB
2450  NEXT
2460  FOR GREEN=ED TO ST+1 STEP -1
2470    GOSUB *CINTSUB
2480  NEXT
2490  FOR RED=ST TO ED-1
2500    GOSUB *CINTSUB
```

```
2510  NEXT
2520  FOR BLUE=ED TO ST+1 STEP -1
2530     GOSUB *CINTSUB
2540  NEXT
2550 RETURN
2560 '
2570 *CINTSUB
2580  COL(NUM,1)=BLUE
2590  COL(NUM,2)=RED
2600  COL(NUM,3)=GREEN
2610  NUM=NUM+1
2620 RETURN
2630 '
2640 *TILEDATA
2650  DATA 0,0,136,34,170,85,238,187,255,255
2660 '
2670 *TRAP
2680  RESUME NEXT
```

このプログラムは，VIEWで設定したスクリーン座標と，WINDOWで設定したワールド座標との値は，完全に一致したものとなっています．したがって，変数X，Yは，各グラフィック命令に対して，ワールド座標系，GET@・PUT@命令に対して，スクリーン座標系の値というように，２つの意味を持っていることになります．では，スクリーン座標とワールド座標の値が，一致していない場合を考えてみましょう．

プログラム24において，ある期間のグラフを表示した後で，Gカーソルによって，各頂点の値を読むことします．この場合，カーソルの位置を示す変数X，Yは，スクリーン座標の値を保持しており，ウィンドウの切りとり方が表示するグラフの範囲によってまちまちなので，ワールド座標とは，一致していません．カーソルを移動させて，その点の降水量を求めようとしても，情報としてスクリーン座標の値しかないのでは，難しいことです．そこで，4.3.8で出てきたMAP関数を用います．降水量は，縦軸の値であり，－１倍して，あるデータを負の値にしてそのままY座標の値にしています．ですから，スクリーン座標上の値を持つYから降水量を求めるには，

PRECIP＝MAP（Y，３）／（−10）

とすれば，よい訳です（MAP関数の機能のパラメータが３の場合の動作は，（表13）を参照して下さい）．

```
100 '
110 '   Program 37
120 '     window demonstration
130 '
140 '
145  GOSUB *CURINIT
150  SCREEN 0,0 : CLS 3
160  WINDOW(-30,-560)-(730,40)
170 '
180 *START
190  CLS 2
200 '
210  LINE(0,-510)-(0,0),7
220  LINE -STEP(730,0),7
230  FOR I=0 TO -500 STEP -50
240    LINE(-5,I)-(0,I),7
250  NEXT
260  FOR I=0 TO 700 STEP 50
```

```
270   LINE(I,-10)-(I,0),7
280  NEXT
290  GOSUB *NUMBER
300 '
310  RESTORE *PRECIPITATION
320  READ A$,P : X=0 : POINT(10,-P)
330 '
340  RESTORE *PRECIPITATION
350  FOR I=1 TO 15
360    READ A$
370    FOR J=1 TO 5
380      READ P : X=X+10
390      LINE -(X,-P),5
400      CIRCLE STEP(0,0),1,3
410    NEXT
420  NEXT
430  LOCATE 0,0 : PRINT TAB(20);"年ﾍｲｷﾝ ｺｳｽｲ ﾘｮｳ (5日 ｵｷ)"
440  PRINT "(mm)";SPC(20)
450  IF INKEY$="" THEN 450
460 '
470  LOCATE 0,0 : PRINT SPC(70)
480  LOCATE 0,0 : INPUT "ﾄﾞﾉ ｷｶﾝ ｦ､ ｺﾏｶｸ ﾐﾀｲ ﾃﾞｽｶ ";N
490  IF N=0 OR N>15 THEN END
500  ON N GOSUB *SET1,*SET2,*SET3,*SET4,*SET5,*SET6,*SET7,*SET8,
               *SET9,*SET10,*SET11,*SET12,*SET13,*SET14,*SET15
510  LINE((N-1)*50+10,-510)-(N*50,0),2,B
520  INPUT "ｺﾉ ﾊﾝｲ ﾃﾞ ｲｲﾃﾞｽｶ ";C$
530  IF C$<>"y" THEN *START
540  CLS 3
550 '
560  READ A$ : LOCATE 20,0
570  PRINT A$;" ﾉ ﾍｲｷﾝ ｺｳｽｲﾘｮｳ"
580  PRINT "(mm)"
590  WINDOW(N*50-10,-560)-(N*50+60,40)
600 '
610  LINE(N*50-5,-510)-(N*50-5,0),7
620  LINE -STEP(50,0),7
630  FOR I=0 TO -500 STEP -50
640    LINE(N*50-5.5,I)-(N*50-5,I),7
650  NEXT
660  FOR I=N*50 TO N*50+40 STEP 10
670    LINE(I,-10)-(I,0),7
680  NEXT
690  GOSUB *NUMBER
700 '
710  READ P : X=N*50 : POINT(X,-P) : CIRCLE STEP(0,0),.2,2
720 '
730  FOR I=2 TO 5
740    READ P : X=X+10
750    LINE -(X,-P),1
760    CIRCLE STEP(0,0),.2,2
770  NEXT
775  GOSUB *MOVE
780  IF INKEY$="" THEN 780
790 END
800 '
810 *NUMBER
820  LOCATE 0,2 : PRINT " 50"
830  LOCATE 0,12 : PRINT " 25"
840  LOCATE 0,23 : PRINT "  0";
850 RETURN
860 '
1000 *SET1  RESTORE *DATA1  : RETURN
1010 *SET2  RESTORE *DATA2  : RETURN
1020 *SET3  RESTORE *DATA3  : RETURN
1030 *SET4  RESTORE *DATA4  : RETURN
1040 *SET5  RESTORE *DATA5  : RETURN
1050 *SET6  RESTORE *DATA6  : RETURN
1060 *SET7  RESTORE *DATA7  : RETURN
1070 *SET8  RESTORE *DATA8  : RETURN
1080 *SET9  RESTORE *DATA9  : RETURN
1090 *SET10 RESTORE *DATA10 : RETURN
1100 *SET11 RESTORE *DATA11 : RETURN
1110 *SET12 RESTORE *DATA12 : RETURN
```

```
1120 *SET13 RESTORE *DATA13 : RETURN
1130 *SET14 RESTORE *DATA14 : RETURN
1140 *SET15 RESTORE *DATA15 : RETURN
1150 '
1160 '
2000 *PRECIPITATION
2010 '
2020 *DATA1  :DATA    1/1-1/25,116, 52, 63, 54, 86
2030 *DATA2  :DATA   1/26-2/19, 75,127,107, 92,115
2040 *DATA3  :DATA   2/20-3/16, 91,182,130,132,160
2050 *DATA4  :DATA   3/17-4/10,172,167,185,179,219
2060 *DATA5  :DATA    4/11-5/5,216,201,203,201,246
2070 *DATA6  :DATA    5/6-5/30,189,235,310,245,213
2080 *DATA7  :DATA   5/31-6/24,251,235,344,348,289
2090 *DATA8  :DATA   6/25-7/19,386,254,270,244,177
2100 *DATA9  :DATA   7/20-8/13,290,150,135,109,172
2110 *DATA10 :DATA    8/14-9/7,179,316,446,315,109
2120 *DATA11 :DATA    9/8-10/2,207,405,320,482,306
2130 *DATA12 :DATA  10/3-10/27,498,419,146,229,256
2140 *DATA13 :DATA 10/28-11/21,281,191, 61,135,227
2150 *DATA14 :DATA 11/22-12/16,187,194,125, 69, 96
2160 *DATA15 :DATA 12/17-12
10000 '
10010 *CURINIT
10020  DIM CU%(75)
10030  '
10040  SCREEN 0,2
10050  VIEW(0,0)-(55,25)
10060  LINE(0,10)-(50,10),7
10070  LINE(25,0)-(25,20),7
10080  GET@(17,6)-(32,14),CU%
10090  SCREEN 0,0
10100 '
10110  RESTORE *STEPDATA
10120  FOR I=1 TO 9
10130    READ DX(I)
10140  NEXT
10150  FOR I=1 TO 9
10160    READ DY(I)
10170  NEXT
10180 '
10190 *STEPDATA
10200  DATA 1,2,4,8,16,32,64,128,256
10210  DATA 1,2,3,4,8,16,32,64,128
10220 '
10230  INC0=1 : INC1=1
10240 RETURN
10250 '
10260 *MOVE
10270  X=320 : Y=100
10280  PUT@(X-8,Y-4),CU%,XOR
10290  GOSUB *GETKEY
10300 '
10310 *LOOP1 : IF CH=13 THEN *EXIT1
10320     IF (CH>&H30) AND (CH<=&H39) THEN
         ID=CH-&H30 : INC0=DX(ID) : INC1=DY(ID) : GOTO *GETNEXT
10330     IF (CH<28) OR (CH>=32) THEN *GETNEXT
10340     PUT@(X-8,Y-4),CU%,XOR
10350     IF CH=29 THEN X=X-INC0 : IF X-8<0 THEN X=627
10360     IF CH=28 THEN X=X+INC0 : IF X-8>619 THEN X=8
10370     IF CH=30 THEN Y=Y-INC1 : IF Y-4<0 THEN Y=193
10380     IF CH=31 THEN Y=Y+INC1 : IF Y-4>189 THEN Y=4
10390     PUT@(X-8,Y-4),CU%,XOR
10400   *GETNEXT GOSUB *GETKEY
10410  GOTO *LOOP1
10420 *EXIT1 : PRECIP=-MAP(Y,3)/10
10430  LOCATE 0,0 : PRINT USING "###.#";PRECIP
10440 RETURN
10450 '
10460 *GETKEY
10470  CH=ASC(INPUT$(1))
10480 RETURN
```

今まで述べた，GET@とPUT@の〝@″は，省略してGET，PUTと書いてもさしつかえありません．

PUT@は，GET@で配列にしまったグラフィック・パターンを表示するだけでなく，オプションROMの中の漢字文字を表示させることができます．表示させたい漢字（文字）は，漢字コードによって指定することができ，そのパターンは16×16ドット（8×8，8×16ドットのものも含まれている）の大きさです．書式は普通のPUT@命令の〝配列変数名　（要素ナンバ）″の部分を〝KANJI（漢字コード）″とするだけでいいのです．したがって，

PUT@(X，Y)，KANJI（漢字コード），条件，フォア・グラウンド・カラー
，バック・グラウンド・カラー

と書きます．

書式を見て分かる通り，表示する位置（座標）は，漢字1文字を出力するたびに，毎回指定しなければなりません．このことは，事務処理などのように漢字文章を扱う場合には，チョットわずらわしくなります．できればオートインクリメント（普通のPRINT文のように，1文字書けば，カーソルが次のカラムに移る）にして欲しかった気がします．

しかし，逆に考えてみれば，座標の指定ができるのですから，画面上の任意の位置に表示できる訳です．字と字の間隔や，行と行の間隔を適当にとって見易くすることもできます，8×8の文字を使えば，添字の表示や，指定表示など細かな位置指定も可能になります．

では，漢字を使ったチョットしたサンプルプログラムをお見せしましょう．ただし，これは漢字ROMを装着しているN_{88}－DISK BASICでなければ実行できません．

このプログラムは，人間が教えたことを単純に反復するもので，BRAIN%という配列変数の中に，人間とN_{88}が対話することにより，漢字コードを並べていって，簡単な辞書のようなものを作成します．その対話は非常に単純で，人間が使える言葉は，次の3つです．

yes, no, end

では，プログラムをRUNさせて下さい．次のメッセージが現れたと思います．

```
ｺｳｶｲｿﾞｳﾄﾞ ﾉ CRT ﾃﾞｽｶ ? ■
```

これに，yes，noで答えて下さい．プロンプトのマーク〝]″が出ているときに，yes，no，end以外の言葉を入力すると，それに対応した漢字（文字）のコードを憶えようとして，コードの入力を促してきます．このときのコード入力は，16進数で行い，各コードはスペースで区切ります．次の写真を見てください．この例のように憶えさせていきます．では，どうやって記憶しているか，参考に（図19）に示してみましょう．

今は，漢字とコードと文字を配列で結びつけていますが，記憶する場所を，配列でなくフロッピーディスクに変えて，ちょっと工夫すれば，日本語のワード・プロセッサも，うまく作れると思います．

このように文を出力させた時は,
最初に登録した文字のみを出力します.
その時は,ここではnoと答えないでください.

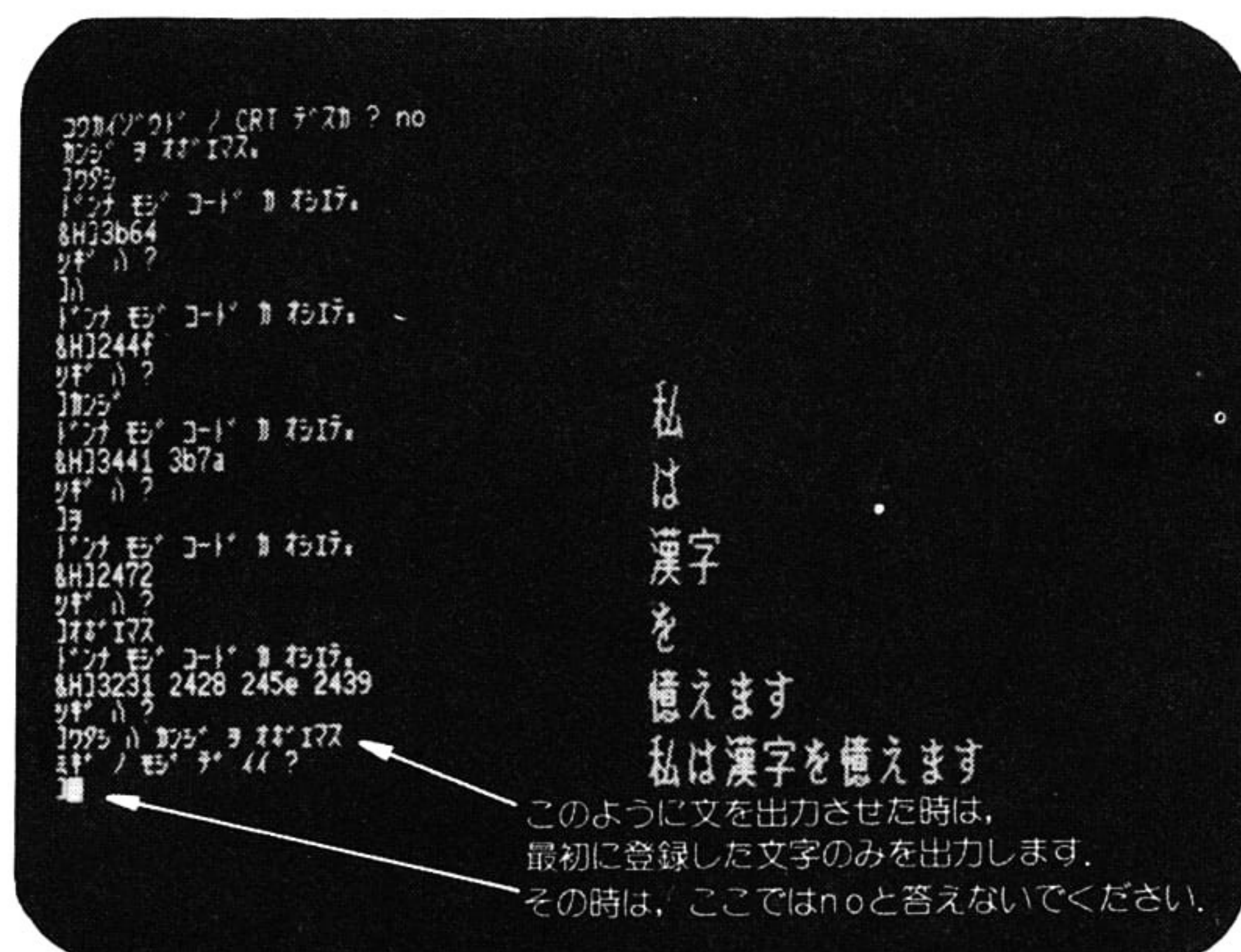



写真 3

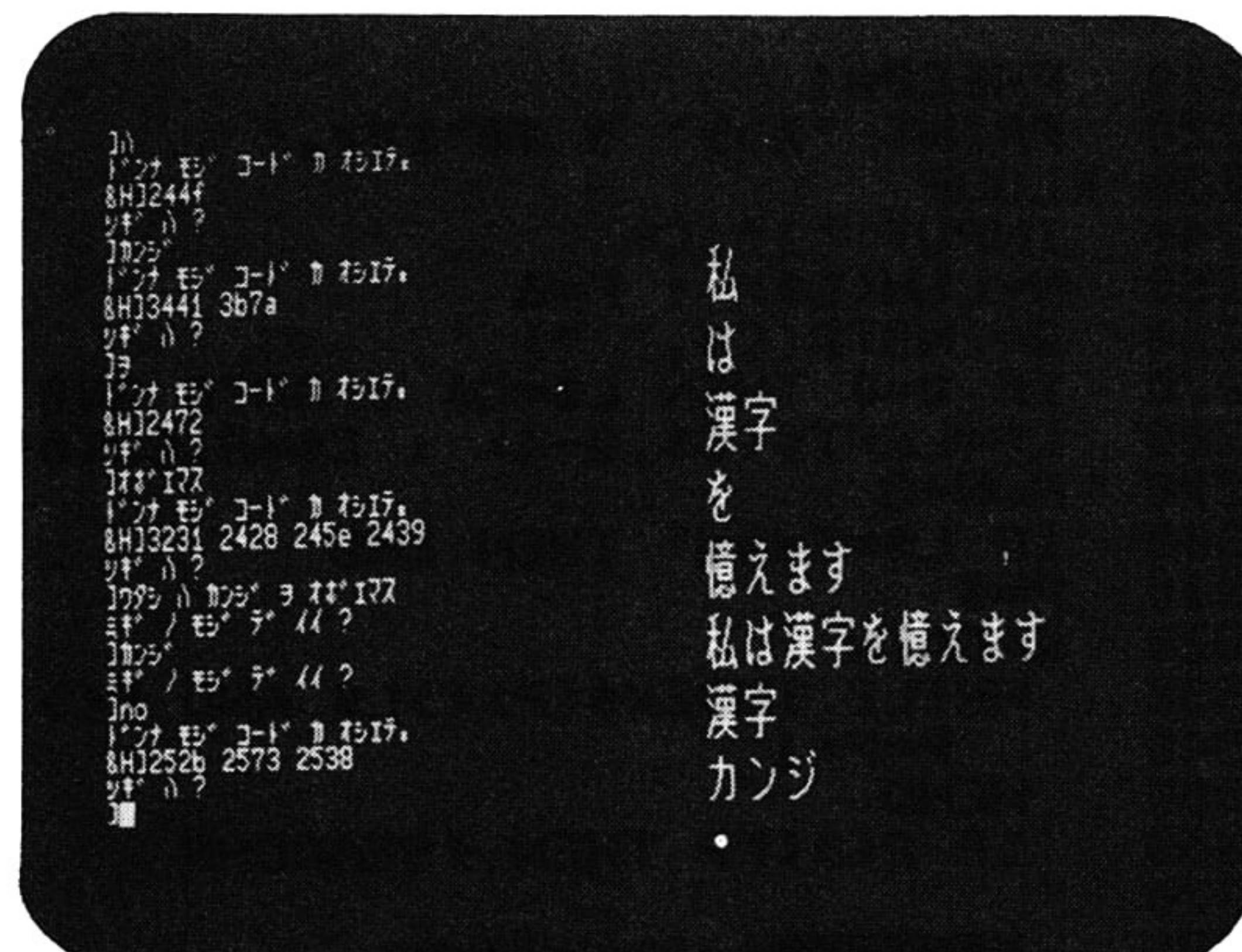



写真 4

キャラクタを登録する.	BRAIN%の登録されている所を指す.	漢字コードを登録する(16進).	
INDEX$(0)=ワタシ	ID%(0)= 1	BRAIN%(0)=0H	
INDEX$(1)=ハ	ID%(1)= 4	BRAIN%(1)=0H	←次へのポインタ
INDEX$(2)=カンジ	ID%(2)= 7	BRAIN%(2)=1H	←漢字のコード数
INDEX$(3)=ヲ	ID%(3)=11	BRAIN%(3)=3B64H	←漢字コード
INDEX$(4)=オボエマス	ID%(4)=14	BRAIN%(4)=0H	
INDEX$(5)=	ID%(5)= 0	BRAIN%(5)=1H	
INDEX$(6)=	ID%(6)= 0	BRAIN%(6)=244FH	
INDEX$(7)=	ID%(7)= 0	BRAIN%(7)=14H	(16進で14は,10進では20となる.)
INDEX$(8)=	ID%(8)= 0	BRAIN%(8)=2H	
INDEX$(9)=	ID%(9)= 0	BRAIN%(9)=3441H	
INDEX$(10)=	ID%(10)= 0	BRAIN%(10)=3B7AH	
INDEX$(11)=	ID%(11)= 0	BRAIN%(11)=0H	
INDEX$(12)=	ID%(12)= 0	BRAIN%(12)=1H	
INDEX$(13)=	ID%(13)= 0	BRAIN%(13)=2472H	
INDEX$(14)=	ID%(14)= 0	BRAIN%(14)=0H	
INDEX$(15)=	ID%(15)= 0	BRAIN%(15)=4H	
INDEX$(16)=	ID%(16)= 0	BRAIN%(16)=3231H	
INDEX$(17)=	ID%(17)= 0	BRAIN%(17)=2428H	
INDEX$(18)=	ID%(18)= 0	BRAIN%(18)=245EH	
INDEX$(19)=	ID%(19)= 0	BRAIN%(19)=2439H	
INDEX$(20)=	ID%(20)= 0	BRAIN%(20)=0H	
INDEX$(21)=	ID%(21)= 0	BRAIN%(21)=3H	
INDEX$(22)=	ID%(22)= 0	BRAIN%(22)=2523H	
INDEX$(23)=	ID%(23)= 0	BRAIN%(23)=2573H	
INDEX$(24)=	ID%(24)= 0	BRAIN%(24)=2538H	
INDEX$(25)=	ID%(25)= 0	BRAIN%(25)=0H	

図19 漢字コードの登録され方

```
100 '
110 '   Kanji demonstration
120 '
130 '
140  CONSOLE 0,25,0
150  SCREEN 2,0 : CLS 3
160  VIEW(320,0)-(639,399)
170  DIM INDEX$(99),ID%(99),BRAIN%(700)
180 '
190  INPUT "ｺｳｶｲｿﾞｳﾄﾞ ﾉ CRT ﾃﾞｽｶ ";A$
200  IF A$="yes" THEN YC=380 ELSE YC=180
```

```
210  FREEP=1
220  PRINT "ｶﾝｼﾞ ｦ ｵﾎﾞｴﾏｽ｡"
230 *LOOP
240  INPUT "]",C$
250  N=LEN(C$) : F=0
260  Q=INSTR(C$," ")
270  IF Q=0 THEN L$=C$ : GOTO 300
280  L$=LEFT$(C$,Q-1) : C$=RIGHT$(C$,N-Q)
290  N=N-Q
300  IF L$="end" THEN END
310  P=0
320  *AGAIN
330   WHILE INDEX$(P)<>L$ AND P<>IFP
340     P=P+1
350   WEND
360   IF P<>IFP THEN *SEARCHR
370  '
380   *ENTRY0
390     INDEX$(IFP)=L$ : IFP=IFP+1
400     GOSUB *RETRY
410     ID%(P)=FREEP : FREEP=FREEP+C
420     *ENTRY1
430       GOSUB *KPRINT
440       PRINT "ﾂｷﾞ ﾊ ?"
450 GOTO *LOOP
460 '
470 *RETRY
480  PRINT "ﾄﾞﾝﾅ ﾓｼﾞ ｺｰﾄﾞ ｶ ｵｼｴﾃ｡"
490  INPUT "&H]",H$
500  N=LEN(H$) : Q=1 : C=2
510  WHILE Q<>0
520     Q=INSTR(H$," ")
530     IF Q=0 THEN G$=H$ : GOTO *ENTRY2
540     G$=LEFT$(H$,Q-1) : H$=RIGHT$(H$,N-Q)
550     N=N-Q
560     *ENTRY2
570       BRAIN%(FREEP+C)=VAL("&H"+G$)
580       IF VAL("&H"+G$)=0 THEN *ESC
590       C=C+1
600       *ESC
610  WEND
620  BRAIN%(FREEP+1)=C-2 : KP=FREEP
630 RETURN
640 '
650 *SEARCHR
660  KP=ID%(P) : BKP=KP : GOSUB *KPRINT
670  IF Q<>0 THEN F=1 : GOTO 260
680  *MORE
690   PRINT "ﾐｷﾞ ﾉ ﾓｼﾞ ﾃﾞ ｲｲ ?"
700   INPUT "]",A$
710   IF A$<>"no" THEN C$=A$ : GOTO 250
720   BKP=KP : KP=BRAIN%(KP)
730   IF KP=0 THEN *DEFF
740   GOSUB *KPRINT
750  GOTO *MORE
760 '
770 *DEFF
780  GOSUB *RETRY
790  BRAIN%(BKP)=FREEP
800  FREEP=FREEP+C : KP=BRAIN%(BKP)
810  GOTO *ENTRY1
820 '
830 *KPRINT
840  C=BRAIN%(KP+1)
850  IF F=0 THEN X=0 : ROLL 20
860  FOR I=1 TO C
870    IF X>19 THEN X=0 : ROLL 20
880    PUT@(X*20,YC),KANJI(BRAIN%(KP+I+1))
890    X=X+1
900  NEXT
910 RETURN
```

改行の処理(20ドット)

4.3.10 WINDOW関数，VIEW関数—便利な関数その2—

今まで話してきた，4.3.6の WINDOW命令や4.3.7のVIEW 命令は，どちらもウィンドウやビューポートの範囲を設定するだけでした．ここで説明するのは，既に設定されているウィンドウやビューポートが，どんな範囲にあるかを知るための関数です．

WINDOW 関数は，文字通り，現在ウィンドウが設定されている範囲を教えてくれます．教えてくれる値は，ウィンドウを設定するときと同様に，ワールド座標における値です．得ることができるのは，4つの値で，4.3.6 のWINDOW命令で設定するパラメータに一致します．つまり，

WINDOW（X1，Y1）－（X2，Y2）

と設定した後に，WINDOW 関数で得られる数値は，X1，Y1，X2，Y2の4つです．値は，4つありますから，1度に全てを得る訳にはいきません．そのために，どの値を必要としているのかを示すパラメータを関数に与えています．

パラメータは，0から3まであります．WINDOW 関数の書式は，POINT 関数のときのように，

WINDOW（パラメータ）

と書きます．パラメータの持つ意味は，(図20)の通りです．すなわち，パラメータに0を与えると，ウィンドウの左上の頂点のX座標，1を与えるとウィンドウの左上の頂点のY座標を返し，これは先程のX1，Y1に当たります．同じように，パラメータに2，3を与えると，それぞれ，ウィンドウの右下の頂点のX座標，Y座標を返してきます．

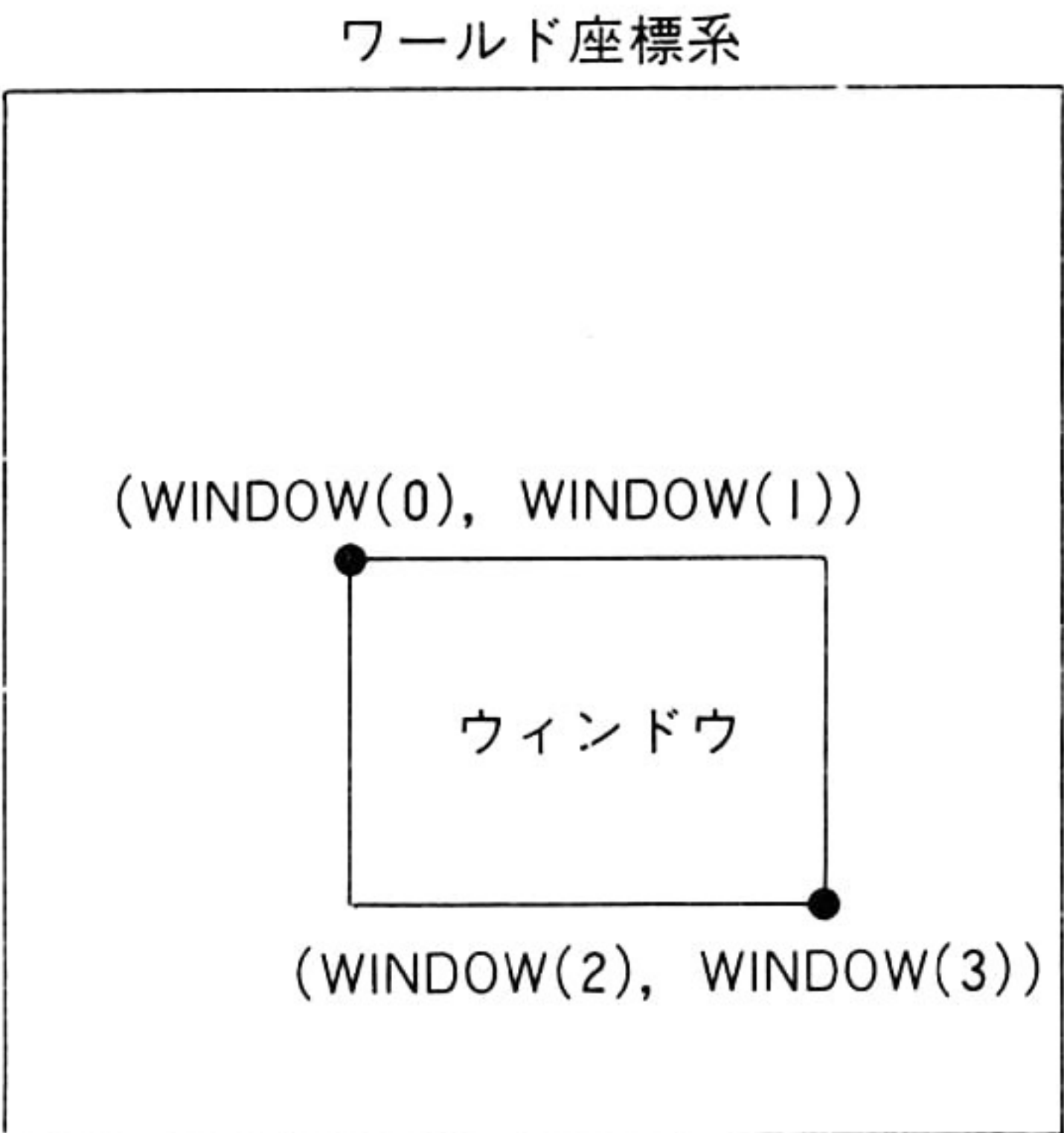



図20　WINDOW関数とパラメータ

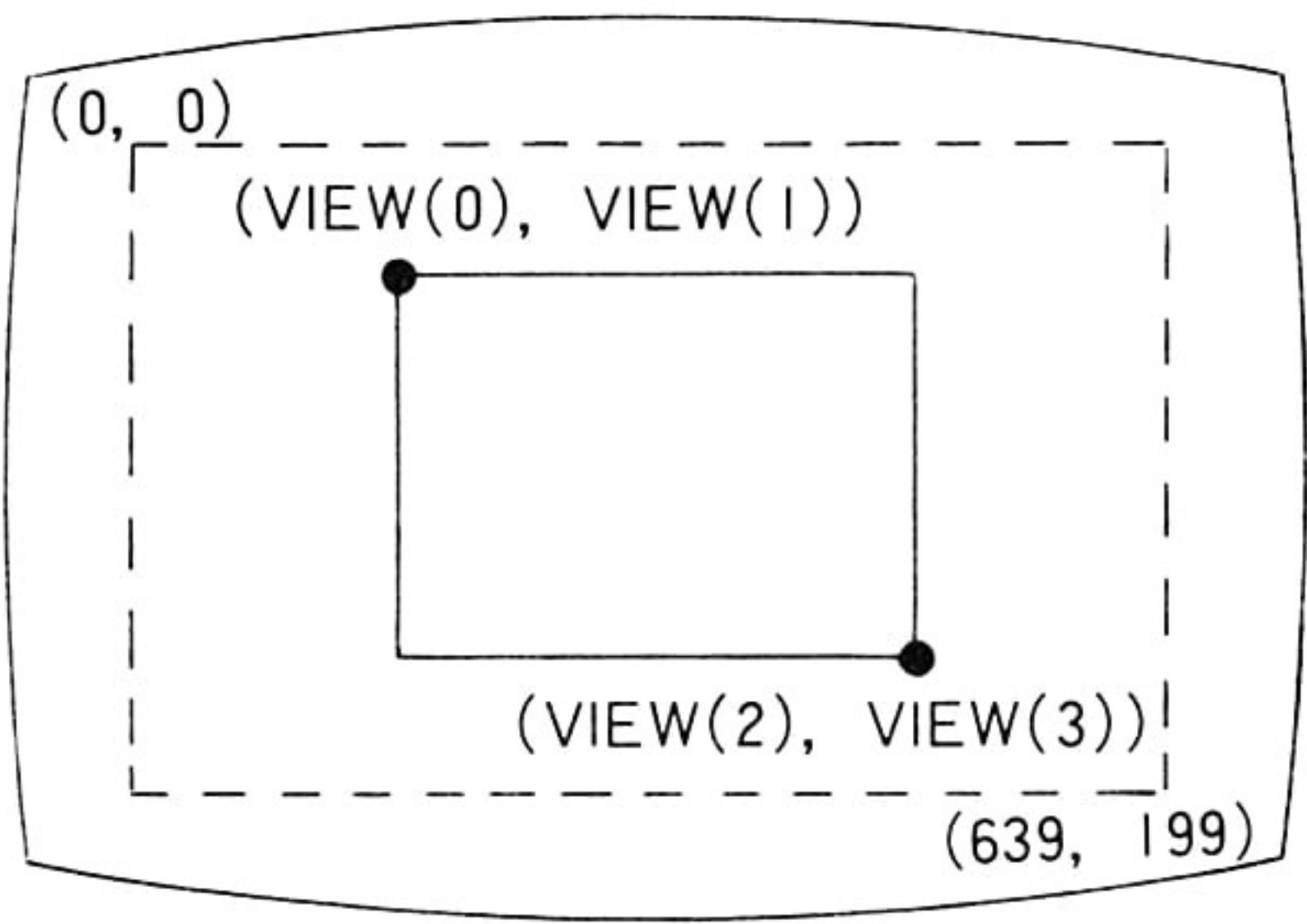



図21　VIEW関数とパラメータ

WINDOW 関数が分かれば，VIEW 関数も同じような働きをするのは見当がつくと思います．VIEW 関数は，現在のビューポートが，どこに設定されているのかを知るために，用意された関数ですが，WINDOW 関数と異なって，教えてくれる値は，ワールド座標における値ではありません．では，いったいどんな値を返してくるのでしょうか．

話は少々それますが，4.3.7の VIEW 命令では，触れなかったことですが『画面のどこに表示させるかを設定する』という言葉は，暗黙のうちに次の制約を持っています．すなわち，

VIEW（X１，Y１）－（X２，Y２）

と設定したときに，各パラメータの取りうる値は，X１，X２が０から639，Y１，Y２が０から199の範囲内に限られます．その上，X１＜X２，Y１＜Y２も満たしている必要があります．したがって，VIEW関数の返す値もある範囲内の数値に限られます．

パラメータは０から３で，それぞれ（図21）のように位置の座標を返してきます．当然，返ってきた値は，

０≦VIEW（０）＜639
０≦VIEW（１）＜199
０＜VIEW（２）≦639
０＜VIEW（３）≦199
VIEW（０）＜VIEW（２）
VIEW（１）＜VIEW（３）

のような，条件を満たしていなければなりません．

では，WINDOW 関数と VIEW 関数を使って，プログラム32のMAP関数のシミュレーションを完全なものにしてみましょう．プログラム32では，ワールド座標からスクリーン座標への変換しか行ってませんが，スクリーン座標からワールド座標への変換も行っているのが，プログラム40です．パラメータの与え方は，

MAP（C，F）

と書くのと同様に，変数のＣとＦに代入してから，＊IMITATIVEMAPを呼び出しています．150行と160行の間で WINDOW とVIEW を適当に設定するプログラムを書きたして下さい．結果は，MAP関数の値とシミュレーションで求めた値の両方が表示されます．＊IMITATIVEMAPが，MAP関数と同じ機能を行っているのが，分かって頂けたと思います．

```
100 '
110 '   Program 40
120 '     MAP simulation
130 '
140 '
150  CLS : LOCATE 30,0 : PRINT "MAP simulation"
160  INPUT "MAP(C,F) : ",C,F
170 '
180  GOSUB *IMITATIVEMAP
190  PRINT "MAP(";C;",";F;") = ";MAP(C,F)
200  PRINT "IMITATION = ";MAPVAL
210 END
220 '
230 *IMITATIVEMAP
240  ON F GOTO *FUNC0,*FUNC2,*FUNC3
```

```
250   IF F<>0 THEN MAPVAL=0 : RETURN
260 '
270 *FUNC0
280   STACKX=POINT(0)  : STACKY=POINT(1)
290   POINT(C,C)
300   MAPVAL=POINT(F+2)
310   POINT(STACKX,STACKY)
320 RETURN
330 '
340 *FUNC2
350   MAPVAL=WINDOW(0)+C*(WINDOW(2)-WINDOW(0))/(VIEW(2)-VIEW(0))
360 RETURN
370 '
380 *FUNC3
390   MAPVAL=WINDOW(1)+C*(WINDOW(3)-WINDOW(1))/(VIEW(3)-VIEW(1))
400 RETURN
```

4.3.11 グラフィック命令のまとめ

これまでに，N88-BASIC の色々なグラフィック命令を見てきましたが，お分かり頂けたでしょうか？

ここで，最初でお約束していたようにグラフィック命令の一覧表を載せておきます．どの命令が，どの画面のどの座標に対して働きかけるものかが，分からなくなった時に活用して下さい．

<table>
<tr><td rowspan="4">画面</td><td></td><td>テキスト画面</td><td colspan="3">グラフィック画面</td></tr>
<tr><td rowspan="3">ステートメント</td><td>CONSOLE</td><td colspan="3">SCREEN</td></tr>
<tr><td colspan="4">COLOR</td></tr>
<tr><td colspan="4">CLS</td></tr>
<tr><td rowspan="5">座標</td><td rowspan="2"></td><td rowspan="2">キャラクタ座標</td><td colspan="3">グラフィック座標</td></tr>
<tr><td>ワールド座標</td><td>スクリーン座標</td><td>その他</td></tr>
<tr><td>ステートメント</td><td>COLOR@
LOCATE</td><td>CIRCLE
LINE
PAINT
POINT
PRESET
PSET
WINDOW</td><td>GET@
PUT@</td><td>VIEW
COLOR
ROLL</td></tr>
<tr><td rowspan="2">関数</td><td rowspan="2">CSRLIN
POS</td><td>WINDOW</td><td>POINT</td><td rowspan="2">VIEW</td></tr>
<tr><td colspan="2">MAP
POINT</td></tr>
</table>

表18 グラフィック命令一覧表

5章　入出力とファイル

5.1　N_{88}-BASICで扱える入出力装置

N_{88}-BASICの特徴の1つに，豊富な周辺装置（入出力装置）を取り扱えることがあります．入出力装置というのは，コンピュータと情報をやりとりするための装置で，人間とコンピュータのコミニュケーションを行うキーボードとスクリーン，外部記憶としてのディスクやカセットなど，様々なものがあります．N_{88}-BASICはこれらの装置を統一的に取り扱うために，ファイルという概念を取り入れています．ここでファイルについて説明をする前に，まずN_{88}-BASICではどんな入出力装置を取り扱えるのかをお話ししましょう．

BASICで取り扱える入出力装置には，次のようなものがあります．

① キーボード
② スクリーン（画面）
③ プリンタ
④ オーディオカセット
⑤ フロッピーディスク
⑥ RS-232Cポート

実にいろいろなものがあります．これらは全て情報(データ)をやりとりするための装置なのですが，その働きによって分類することができます．その分類の方法は次の2通りでしょう．

① **入力装置と出力装置**
② **外部記憶装置とそれ以外の入出力の装置**

入力装置はデータを外部から読み込むもので，出力装置はデータを外部に出力するものです．キーボードは入力装置，スクリーンは出力装置というわけです．フロッピーディスクは入力，出力どちらもできる装置です．

もう１つの分け方は，外部記憶装置であるかそれ以外のものかということです．外部記憶装置は，データをコンピュータの外部に記憶しておくためのものです．記憶装置ですから，当然データを入力・出力する両方の働きがあります．それに対して記憶装置以外のものは，データを単に外部とやりとりするだけのもので，入力か出力のどちらかの働きしかありません（ただし RS-232C ポートの場合は，データの流れが双方向であるために，入力と出力の両方の働きがあります）．

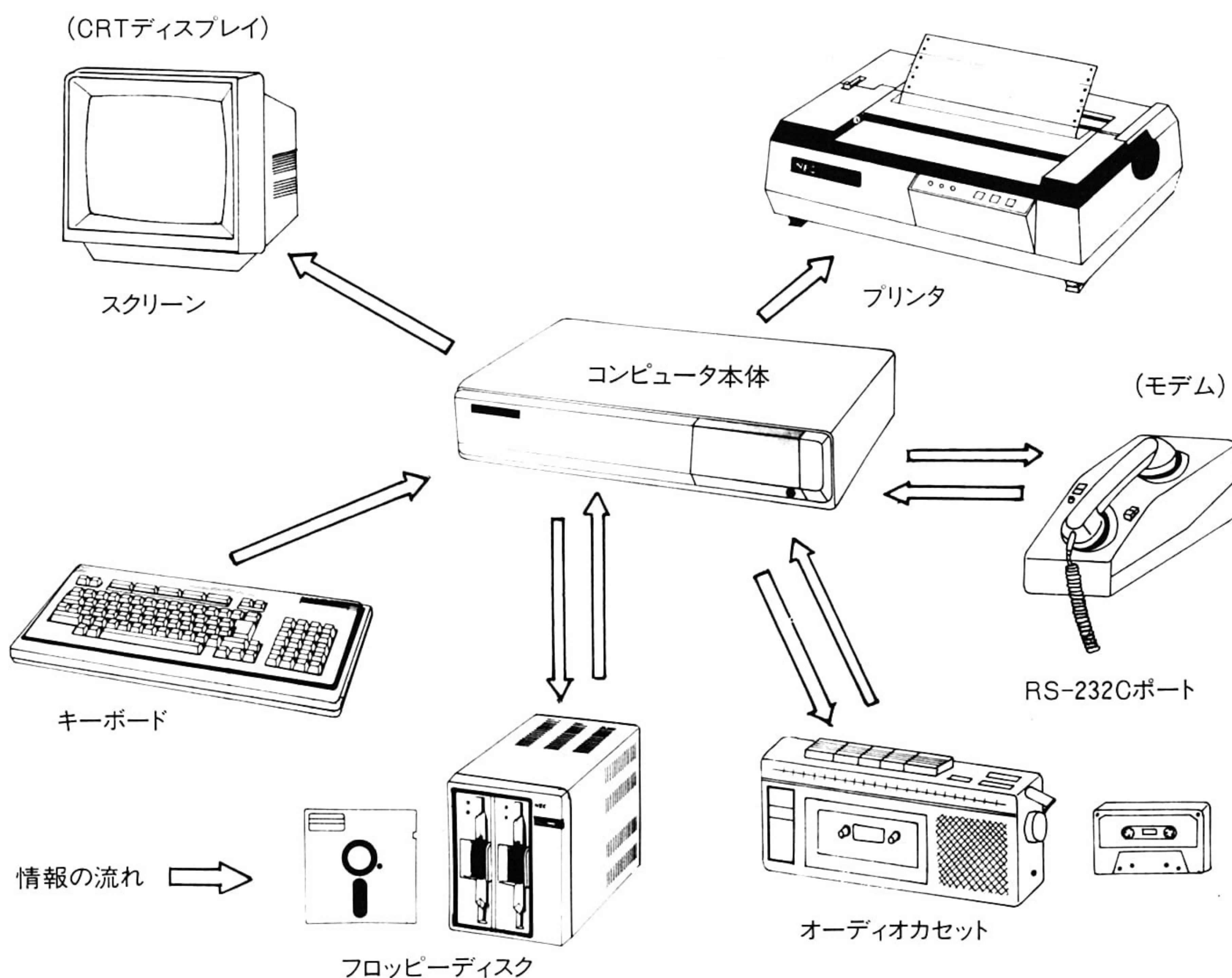



図１　外部装置と情報の流れ

それでは各入出力機器の働きをそれぞれ説明していきましょう．

5.1.1　キーボード

キーボードはオペレータがコンピュータに情報（データ，命令，プログラムなど）を入力するために，最も頻繁に使われる装置です．キーを押すことによって，文字や記号などが入力できます．このキーボードに対して入力を行うには，BASIC の INPUT 命令を使うことはお分かりですね．その他にも LINE INPUT, INPUT$ などの命令（７章を参照）が使えます．

5.1.2　スクリーン

スクリーンとは画面のことで，コンピュータがオペレータに情報を表示するための装置であり，キーボードと組み合わせて最も頻繁に使われる出力装置でもあります．

BASIC のコマンドレベルでの操作においては，このキーボードとスクリーンが入出力を行う装

置と通常は決められているのです．このスクリーンに対して文字や記号などの表示（出力）を行うには，PRINT 命令を使うことができます．

5.1.3 プリンタ

プリンタはコンピュータから出力される情報を紙に印刷する装置です．コンピュータが情報を表示するための装置は，このプリンタとスクリーンの2つであり，どちらかを選んで表示させることができるように，ほぼ同じ程度の命令が使えるようになっています．プリンタに表示を行うには，スクリーンの PRINT に対応する LPRINT 命令，LIST に対する LLIST 命令など，いくつか使えるようになっています．

5.1.4 オーディオカセット

オーディオカセットは，プログラムやデータを外部に出力して保存し，再び入力できる働きを持った外部記憶装置です．コンピュータは内部にメモリを持っており，その中にプログラムやデータを蓄えています．しかし，それらは電源を切ったりすると消えてしまうので，外部に記憶して保存し，後で再び内部に読み込むことができるようにするのが，外部記憶装置なのです．オーディオカセットはパソコンの外部記憶装置として最もポピュラーなもので，装置が安い，手軽であるなどの利点がありますが，記録スピードが遅い，テレコの操作がわずらわしいなどの欠点もあります．BASIC には，このカセットの入出力を行うための命令が用意されています．

5.1.5 フロッピーディスク

フロッピーディスクは，外部記憶装置として最も汎用性に富んだ装置です．ここではフロッピーディスクの仕組みについてはお話ししませんが，記憶容量が大きい，処理速度が速い，ランダムアクセスができるなど，オーディオカセットには無い利点を持っています（ランダムアクセスとは目的のデータをすぐに取り出せる機能です)．ディスクの入出力操作を行う命令も BASIC に用意されています．

5.1.6 RS-232Cポート

RS-232C ポートは，内蔵された RS-232C インターフェースに外部機器を接続して，外部とのデータ通信を行うためのものです．

RS-232C とは，インターフェース規格の1つの標準で，直列（シリアル）データの送受信ができ，このインターフェースを備えた装置とならばほとんどの場合接続することができ、データ交換が可能という便利なものです．N_{88} -BASIC には，この RS-232C ポートを介してデータの入出力を行う命令が用意されているのです．

5.2 使いたい装置を指定する

さて，BASICが扱える入出力装置について述べてきましたが，それでは，実際にどうやって入出力を行わせればよいのでしょうか．入出力を行うには，まずその操作を行う装置を指定しなくてはなりません．この装置の指定には次の2つの方法が考えられます．

1つは，各装置ごとに入出力の命令を用意する方法です．画面に出力を行うには，PRINT 命令を使い，プリンタの場合には，LPRINT 命令を使うといった方法です．この考え方からいくと，それ以外の装置に対しても全く別の命令を用意しなければならないことになります．各装置に対してそれぞれ命令が用意されていれば確かに分かり易く便利なのですが，入出力の汎用性が薄れてしまいます．つまり入出力方法（命令）がそれぞれ違っているため，1つの入力装置を他の入力装置に変えるということは簡単にはできません．また入出力の取り扱いも，命令が多いため繁雑になりがちなのです．さらにいうなら，入出力装置それぞれに命令があるわけで，入出力のシステムとして統一性に欠けていると言えるでしょう．これは入出力装置が増える度に，そのための命令の追加を重ねて来たためで，それもそろそろ限界にきている訳です．

そこでもう1つの方法は，入出力を行う前に目的の装置の指定などの一定の手続きをすれば，どんな装置も同じ方法でデータの入出力ができるようにするやり方です．これは大型計算機に使われている考え方で，多種多様な入出力装置をいかに効率よく取り扱うかを目的に考えられたのです．こうすれば入出力は汎用性に富んだものとなり，ユーザーが入出力装置の指定を自由に行うこともできるようになります．

その入出力の考え方を統一したのが，ファイルという概念なのです．

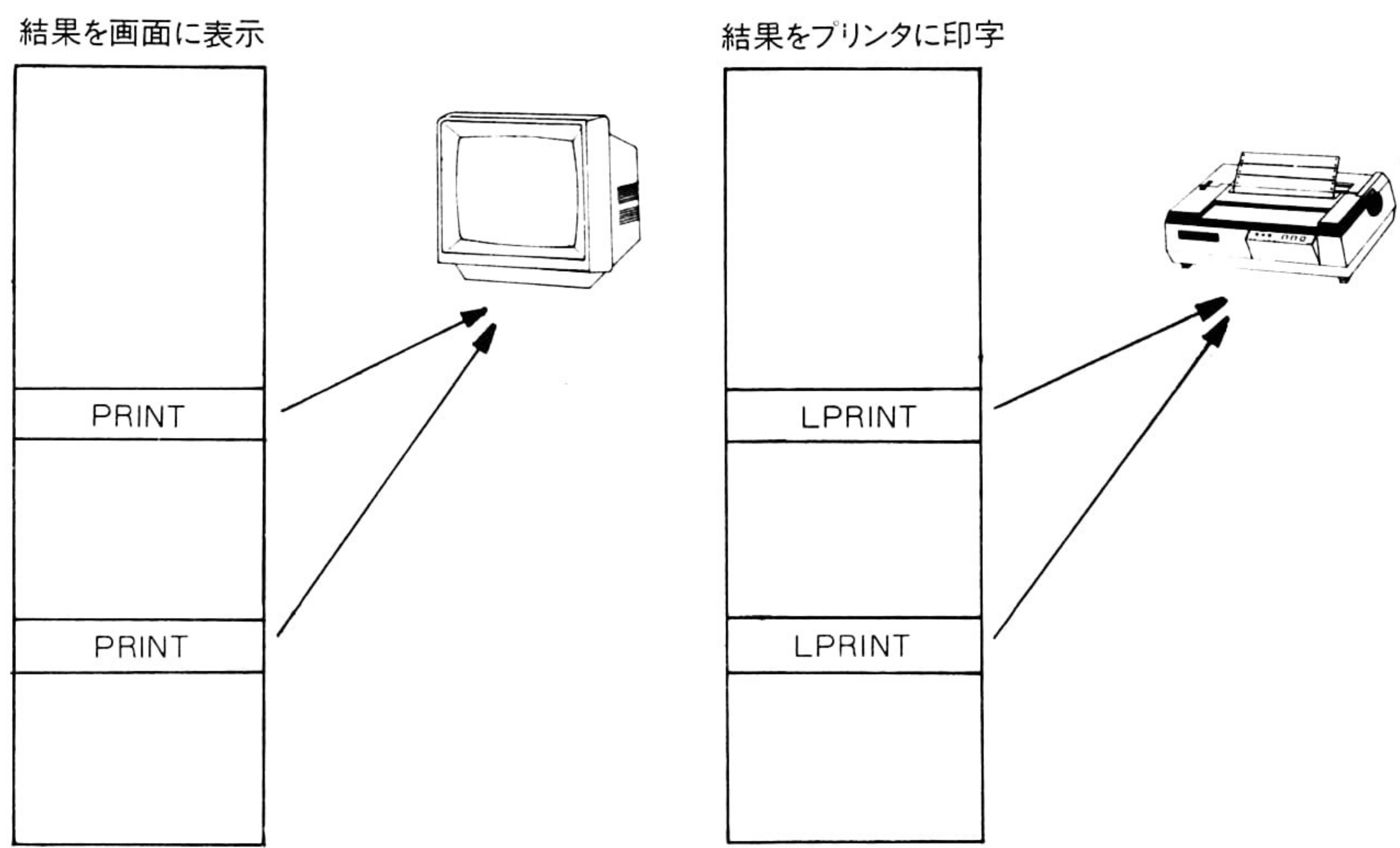



図2　同じ仕事をするプログラムにおいて①出力装置ごとに命令が違う場合

5.3 ファイルとは

皆さんがディスクやカセットテープにプログラムをセーブ・ロードする時のことを考えて下さい．たとえば，「 SAMPLE 」というプログラムをディスクにセーブするとします．その時は次のようにするでしょう．

SAVE "SAMPLE" (ドライブ1にセーブ)

またロードする時は次のように入力します．

LOAD " SAMPLE "

これは，私たちがプログラムというデータを「SAMPLE」という名前で扱っていた訳です．この名前をファイル名と呼びます．ファイル名はファイルに付けた名前ですから，私達はこのプログラムをファイルとして扱っていたのです．

ファイルとは「意味を持ったデータ・情報の集まり」です．プログラムは BASIC の命令の集まりですから，これをファイルとして取り扱ったのです．BASIC は入出力装置と情報をやりとりする場合は，全てこのファイルを1つの単位として扱います．何故ファイルを単位として扱うかと言いますと，入出力の効率を上げるためなのです．入出力装置とデータをやりとりする場合，一度にまとめて転送した方が能率がいいのです．ですから，データをファイルとして扱うと都合

出力に画面を指定

出力にプリンタを指定

PRINT #

PRINT #

PRINT #は指定された装置に出力を行う命令

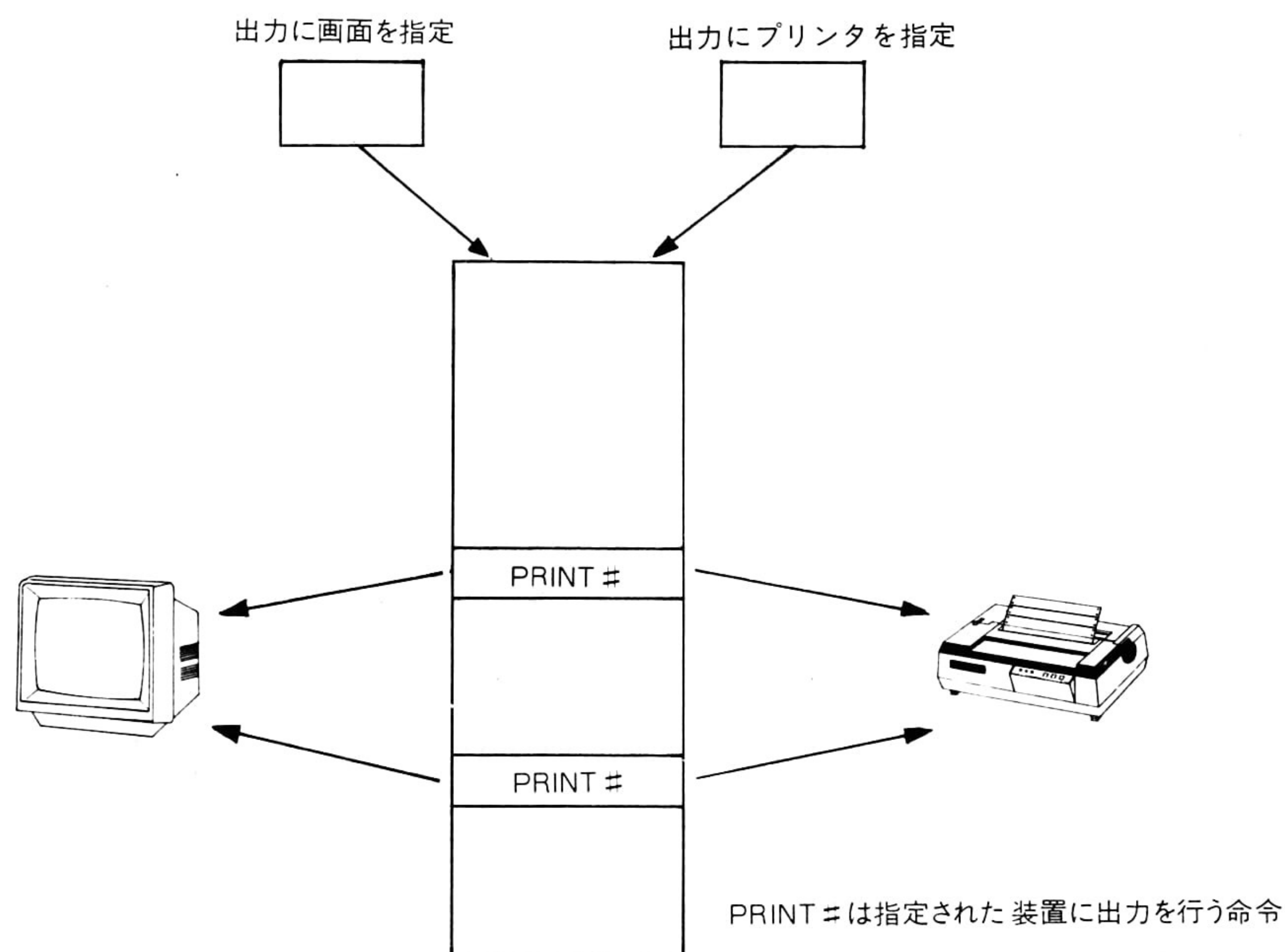
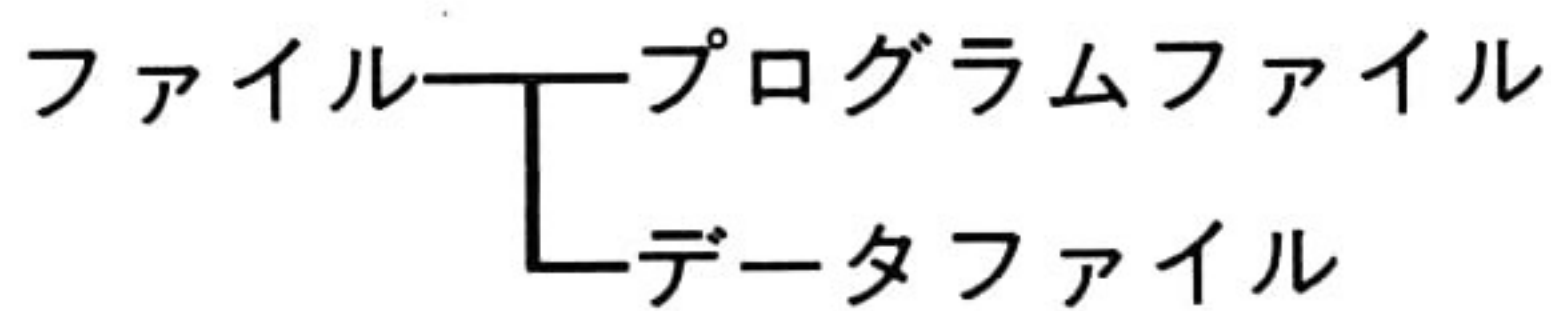



図3　同じ仕事をするプログラムにおいて②出力命令は1つで，前処理によって，出力装置を指定する場合

がいいという訳です．

BASICで扱うファイルは，大きく2つに分けることができます．それはプログラムファイルとデータファイルです．プログラムファイルは，中身がBASICのプログラムであるファイルです．もう1つのデータファイルは，プログラムの実行などによって作られたデータが入っているファイルです．BASICにおいてファイルとは，この2種類のファイルをまとめた呼び方なのです．

ファイル┬プログラムファイル
**　　　　└データファイル**

データを外部記憶装置に記憶するには，ファイル単位で行いますが，ファイル名はそのファイルを他のファイルと区別するために付けるものです．外部記憶装置にはたくさんのファイルを記憶することができます．そのそれぞれのファイルを全く別々のものとして取り扱うためには，それぞれに別の名前を付けなくてはならないことになります．ファイル名はそのファイルを私達が指定するために使いますが，外部記憶装置に記憶されたファイルを区別するためでもあるのです．

ファイル名はファイル同士を区別するためのものなのですが，もっと大きな意味があります．私達はファイルを指定する場合はファイル名を使います．このことをよく考えてみて下さい．ファイル名だけ指定すればよいのです．後は全部BASICが仕事をしてくれ，目的のファイルが外部記憶装置のどこにあろうが，ちゃんとそれを取り出してきてくれる訳です．この働きは変数や配列のそれに似たところがあります．私達はファイルの名前だけを憶えておけばよいのです．

これまでのファイルの話は外部記憶装置でのファイルについてのものでした．それでは外部記憶装置以外のキーボードやプリンタは，どのようなファイルの取り扱いになっているのでしょうか．

これらの入出力装置はデータの流れる方向は常に一定で，単なる入力，出力の働きをするものです．このデータの出入口をチャネルと呼びます．チャネルとはデータの通り道のことで，キーボード，プリンタ，スクリーンなどそれぞれが1つのチャネルというわけなのです．そして入出力装置の指定をこのチャネル（データの通り道）を指定することによって行うようにしたのです．

このチャネルの入出力の考え方を，ファイルの概念を拡張することによって含めて，全ての入出力機器の入出力操作を統一した訳です．

また外部記憶装置とファイルを入出力するには，データを一時蓄えておくための場所が必要です．これをバッファと呼びますが，チャネルによる入出力も同じバッファを介して行うようにして，ファイルと同じ入出力操作ができるようになっています．

入出力装置の指定は，ファイルやチャネルのそれぞれにバッファを割り当て，そのバッファに番号を付けることです．その番号を使って目的のチャネルやファイルに入出力操作を行うのです．この番号は外部記憶装置のファイルに対してはファイル番号，チャネル（入出力機器）に対してはチャネル番号と呼ばれます．どちらも入出力のバッファに付けた番号です．この番号によって，そのバッファを割り当て，どのファイルやチャネルに入出力を行うかを指定するのです．

これによってファイルとチャネルの区別なく入出力が行えるようになります（これ以後の説明においても特別な場合を除き，ファイルとチャネルを区別せず，全てファイルとして説明していきます）.

このファイルとバッファを結びつけて，そのファイルを入出力可能な状態にすることを，ファイルをオープンする（ファイルを開く）といいます.

コンピュータ本体　　外部機器

バッファ

CPU

バッファにはファイル（チャンネル）番号が付けられる.

バッファと外部機器（ファイル）とを結びつけることを，ファイルを開く（オープンする）と言う.

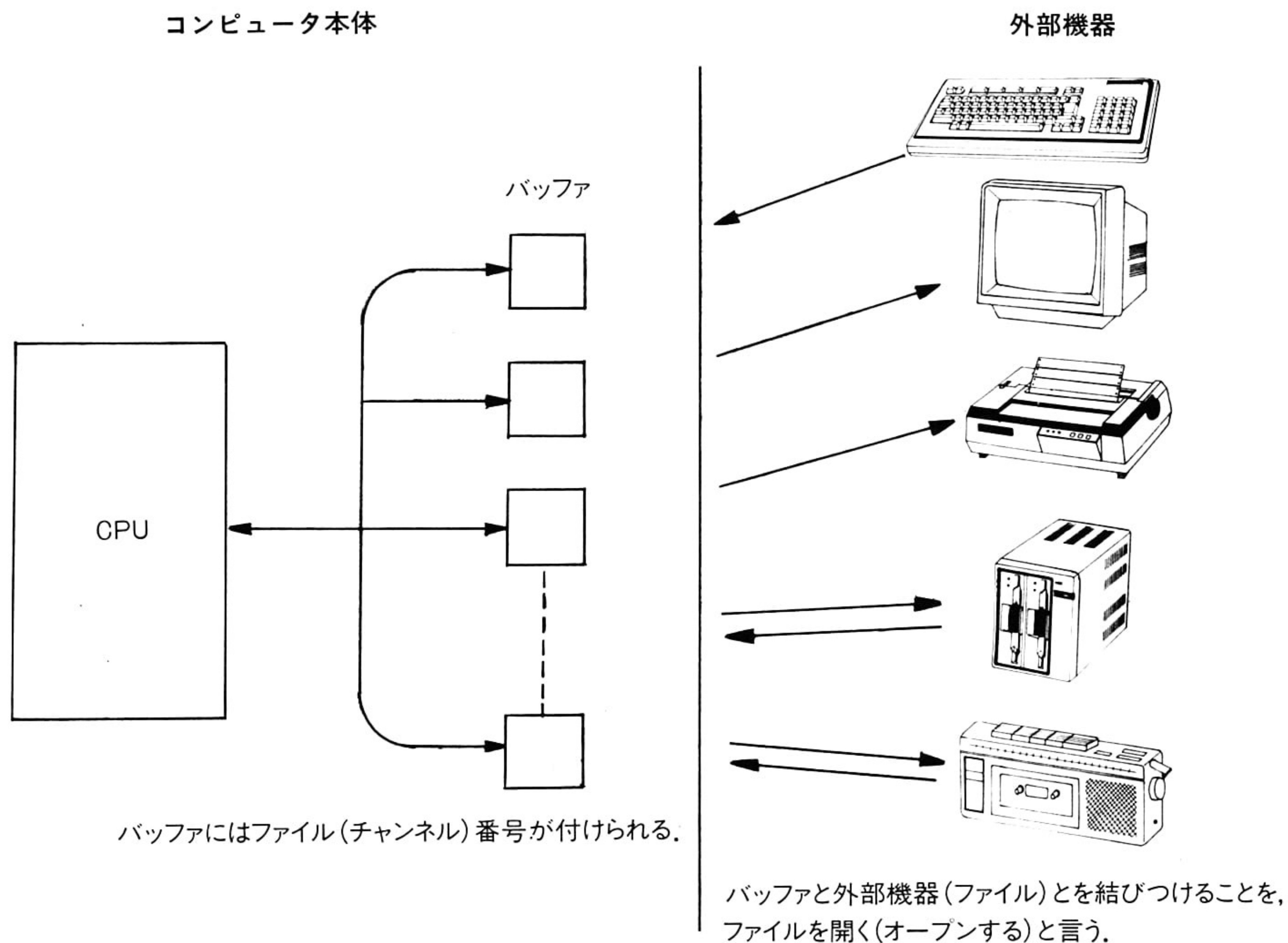



図4　バッファと外部機器（ファイル）

5.4 ファイルディスクリプタによる区別

N_{88}-BASICでは，全ての入出力機器の入出力操作が統一されていることは，前に述べましたが，扱うファイルがどの入出力機器のどのファイルであるかを区別するものが，ファイルディスクリプタと呼ばれるものです．ファイルディスクリプタはLOAD，SAVEにおいてファイルを指定したり，入出力のために開くファイルを指定する場合に用います．

5.4.1 ファイルディスクリプタの実体

ファイルディスクリプタは，次の様な構成の文字列です．

① 〝デバイス名：ファイル名〟

② 〝デバイス名：オプション〟

③　〝デバイス名：〟

④　〝ファイル名〟

ファイルディスクリプタは通常は引用符〝で囲まれた文字列で表しますが，文字変数や文字式などによっても表わすこともできます．また上に示すように，デバイス名，ファイル名は場合によって省略することができます(オプションは必要に応じて付けるようになっています)．デバイス名が省略できるのは，DISK BASICにおいてフロッピーディスク1（〝1：〟)を指定する場合と，ROM BASICにおいてオーディオカセット1（〝CAS1：〟）を指定する場合です．

5.4.2　デバイス固有の名前

N_{88}-BASICで扱える入出力機器の名称を表すのがデバイス名です．デバイス名は入出力機器を指定するためのものです．複数の外部記憶装置（フロッピーディスク）がある場合，目的のファイルが存在する装置の指定に使われます．

デバイス名の後には，必ずコロン：を付けなければなりません．というよりコロンもデバイス名の一部と考えた方がいいでしょう．またデバイス名は大文字でも小文字でもかまいません．デバイス名に含まれる数字が1の場合は省略することができます．

デバイス名	機械名称	入力	出力	備　　考
LPT1： またはLPT:	プリンタ	×	○	
SCRN：	スクリーン	×	○	N_{88}-Disk BASICで使用可
COM1： またはCOM:	RS-232Cポート1	○	○	
CAS1： またはCAS:	カセットテープ1	○	○	転送速度1200ボー
CAS2：	カセットテープ2	○	○	転送速度600ボー
1：	フロッピーディスク1	○	○	N_{88}-Disk BASICでのデフォルト．
2：	フロッピーディスク2	○	○	
3：	フロッピーディスク3	○	○	
4：	フロッピーディスク4	○	○	
5：	フロッピーディスク5	○	○	
6：	フロッピーディスク6	○	○	
7：	フロッピーディスク7	○	○	
8：	フロッピーディスク8	○	○	
KYBD：	キーボード	○	×	N_{88}-Disk BASICで使用可．

表1　デバイス表

5.4.3 ファイル名は9文字以内で

ファイル名とは，プログラムファイルやデータファイルに付ける名前です．ユーザーはこのファイル名によってファイルを取り扱います．ファイル名が必要なのは，外部記憶装置（フロッピーディスク，オーディオカセット）と入出力を行う場合です．その他の入出力機器をデバイスに指定した場合は，ファイル名は省略することができます．

ファイル名の書式は次のようなものです．

ファイル名．拡張ファイル名

書式の中のファイル名は6文字まで，ピリオドの後の拡張ファイル名は3文字までの文字列で表します．もしそれぞれの文字数がそれ以下の場合は，それに満たない部分は空白となります．また拡張ファイル名は省略でき，その場合はピリオドは必要ありません．

ファイル名はファイルの名前を表し，拡張ファイル名はそのファイルの性質を表すものとして使われるのが一般的ですが，拡張ファイル名は自由に使うことができます．

最初のファイル名が，6文字を越えて指定された場合は，先頭から6文字がファイル名となり，それに続く9文字までの3文字が拡張ファイル名となります．なおファイル名の中にコロンとキャラクタコードの0と255の文字は，含むことはできません．また英字の小文字と大文字は別のものとして扱われます．

ファイル名の例を示します．

TEST. 002
ジュウショロク

5.4.4 オプションとして

オプションは，デバイスにRS-232Cポートを指定した時のみ意味を持ち，RS-232Cポートを使用する際のモードやボーレートなどの指定に使われます．

5.4.5 各ファイルディスクリプタにはこんな周辺装置

最後に，各周辺装置を表すファイルディスクリプタの一覧表とその記法の例を示しておきます．

"TEST. 001"	ドライブ1のファイル「TEST. 001」
"2：ジュウショロク"	ドライブ2のファイル「ジュウショロク」
"LPT ："	プリンタの指定（LPT：は LPT1：の省略形）
"CAS2：TEST1"	オーディオカセット（600ボー）のファイル「TEST1」
"SCRN："	スクリーンの指定

5.5 ファイルを操作する

今までは，ファイルの考え方，周辺装置の指定の仕方などの基本的なことをお話しして来ました．これからは，実際のファイルの操作とそのための命令について説明していきます．

今までに出て来たファイルは，プログラムファイルとデータファイルがありました．これから説明していく命令もこれらのファイルを操作するものです．

5.5.1 プログラムをファイルする（SAVE）

SAVEは，ファイルディスクリプタで表されるファイルに，プログラムを書き込む（セーブ）する働きをします．SAVE命令には次の3通りの書式があります．

① SAVE ファイルディスクリプタ
② SAVE ファイルディスクリプタ，A
③ SAVE ファイルディスクリプタ，P

①の方法を内部表現（バイナリ）形式，②の方法をアスキー形式でのセーブと呼んでいます．③の方法はプログラムのプロテクトを行うセーブの方法です．

BASICのプログラムは，一般にリストの形のままではなく，特殊な内部表現の形でメモリ上に蓄えられているのです．①の内部表現形式というのは，プログラムをメモリ上の内部表現のイメージのままでセーブすることです．内部表現のプログラムは元のリストより短くなっており，セーブの時も外部の記憶領域が少なくてすむようになっています．

一方，アスキー形式というのは，プログラムをリストをとった場合と全く同じ形でセーブすることです．前の内部表現形式に比べて必要な記憶領域は多少大きくなります．しかし，アスキー形式のプログラムファイルは，全て文字列でできているわけで，これをデータファイルとしてファイル操作を行うことができるのです．そのほかにも，MERGE命令を使ってプログラムをマージする場合には，アスキー形式のプログラムファイルが必要です．

③のプログラムのプロテクトを行うセーブの方法は，プログラムを，読み出し，書き込み，変更などの操作から保護するためのものです．この方法でプログラムをセーブすると，そのプログラムファイルは，ファイル名の変更や削除は全くできなくなります．またそのプログラムをロードし，実行させることはできますが，プログラムのリストを見たり，変更することはできません．LISTやEDITを行うとエラーとなります．また，モニタに入ることも，メモリを直接読み書きすることも，禁止されており，およそリストを見たり，変更したりするのに考えられるすべての命令を実行不可能にしています．この保護は，1度指定すると解除することはできませんから，間違って行わないよう十分注意して下さい．この方法を使うのは，あくまで完璧なプログラムができあがって，以後絶対変更しないと決めた時でしょう．

ところで，スクリーンやプリンタにもアスキー形式でセーブを行うことができます．チョット変に聞こえますが，結果はそのデバイスにリストを出力したのと同じになります．

5.5.2 プログラムファイルを読み込む(LOAD)

LOADは，ファイルディスクリプタで指定されるプログラムファイルから，プログラムをロードする働きがあります．

これには，2通りの書き方があります．

① LOAD　ファイルディスクリプタ
② LOAD　ファイルディスクリプタ，R

①の場合は，目的のプログラムをロードした後，コマンドレベルに戻ります．一方，②の場合は，プログラムをロードした後，ただちにそのプログラムを実行します．その場合，もしロードする前に開いているファイルがあったとしてもそのままの状態で，クローズされることはありません．

SAVE命令で，内部表現形式やアスキー形式でセーブしたものも，どちらも変わりなくロードできます．

5.5.3 LOAD & GO (RUN)

ファイルディスクリプタで指定されるファイルからプログラムをロードした後，その実行を開始します．

次の2つの書き方があります．

① RUN　ファイルディスクリプタ
② RUN　ファイルディスクリプタ，R

①の場合は，開いているファイルは全て閉じ，メモリもクリアされた後，プログラムがロードされます．②の場合は，開いているファイルはそのままです．これは前に出てきた，

LOAD　ファイルディスクリプタ，R

と全く同じ働きをします．

これらのRオプションを付けた命令は，プログラムのチェインを行うためのものです．プログラムのチェインとは，大きなプログラムをいくつかの独立したプログラムモジュールに分けて，ファイルとし，その各ファイルが，順に読み出されて実行されるようにしたものです．各モジュールの終わりにRUN，またはLOADのRオプション付きのものを用いることによって，順に次のモジュールに実行が移るようにしておくのです．

RUN "file 1"

file 1 RUN"file2", R	→	file 2 RUN"file3", R	→	file 3 END

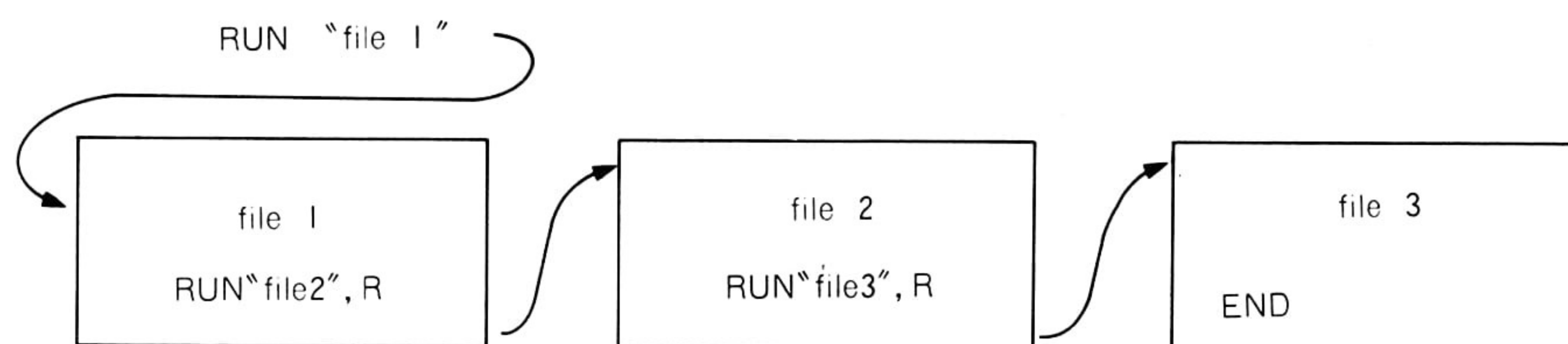



図5　プログラムの連結

ただし，この方法でのチェインは，各モジュール間でデータの受け渡しを行うには工夫が必要です．それぞれのモジュールをロードするごとに，プログラムや変数は全てクリアされてしまうからです．

プログラムのチェインには後で述べる CHAIN 命令を使うと便利です．

5.5.4　ディスケットの中味をのぞく（FILES／LFILES）

FILES はフロッピーディスク専用の命令で，ディスクにどんなファイルが入っているかを表示します．使い方は FILES の後に目的のドライブ番号を付けて入力します．

FILES　ドライブ番号

ドライブ番号の指定が省略された場合は，ドライブ1とみなされます．実際に FILES を行った例を示します．

```
files
backup.n88 2    format.n88 2    setinf.n88 1    xfiles.n88 1    Quartz.    1
Cpaint.    1    DEMO   .   1    DEMOUT.    6    text01 asc 1    sample*bin 1

Ok
■
```

ファイルの表示は，ファイル名と拡張ファイル名，そのファイルの大きさが示されます．ファイル名と拡張ファイル名との間には1文字の空白が取られ，そこにはファイルの形式が示されます．それが空白の場合は，そのファイルは，アスキー形式でセーブされたプログラムファイルであるか，あるいは BASIC のデータファイルであることを示しています．それがピリオドの場合は，内部表現形式でセーブされたプログラムファイルであることを示し，アスタリスクの場合は，後述する BSAVE によって作られた機械語ファイルであることを示しています．

ファイル名の右に表示される数字は，そのファイルの大きさを示しています．この数字の単位はクラスタと呼ばれるもので，ミニフロッピーの場合は2Kバイト，標準の8インチフロッピーでは約6.5Kバイトです．

LFILES は FILES と同じ働きをしますが，出力はプリンタに対して行います．

5.5.5　ファイル名を抹消する（KILL）

KILL はフロッピーディスク専用の命令で，ファイルディスクリプタで指定されるファイルを

ディスクから削除します．書式は次の通りです．

KILL　ファイルディスクリプタ

KILLで削除できるファイルは，プログラムファイル，データファイルなどの全てのディスクファイルです．削除するファイルは，必ず閉じられた状態（現在使われていない状態）でなければなりません．また書き込み禁止の属性（ SET を参照）の付いたファイルは，削除することができません．

KILLでファイルの削除を行う場合，ファイルディスクリプタの指定には，十分注意しなければなりません．もし，ドライブの1と2に同じ名前のファイルがあった時，ドライブ2のファイルを削除したい場合にドライブの指定を忘れると(省略すると)，ドライブ1のファイルの方が削除されてしまいます．またファイル名を間違えた場合に，その間違った名前のファイルがあるとそれが削除されてしまいます．KILL はファイルディスクリプタで指定されたファイルがあると，すぐそれを削除しようとしますから注意が必要です．

```
files 2
backup.n88 2    format.n88 2    setinf.n88 1    xfiles.n88 1    Quartz.    1
Cpaint.    1    DEMO   .   1    DEMOUT.    6    text01 asc 1    sample*bin 1
test   asc 1
Ok
kill "2:test.asc"
Ok
files 2
backup.n88 2    format.n88 2    stinf.n88 1     xfiles.n88 1    Quartz.    1
Cpaint.    1    DEMO   .   1    DEMOUT.    6    text01 asc 1    sample*bin 1

Ok
■
```

5.5.6 ファイル名を付け換える(NAME)

NAME は，ディスクファイルの名前をつけ変える命令です．NAME は次のように用います．

NAME　旧ファイル名　AS　新しいファイル名

どちらのファイル名の指定も，ファイルディスクリプタによって行います．たとえば，ドライブ2の ABC というファイルを EFG に変えたい時は， NAME 〝2： ABC〞 AS 〝2： EFG〞とします．ドライブが1の場合は，ドライブの指定を省略できます．なお，変更を行うファイルは，閉じられた状態でなければなりません．

名前の変更は，当然ですが異なるドライブに渡って行うことはできません．つまり次のようにはできないのです．

NAME　〝ABC〞　AS　〝2：EFG〞

また，新しく付ける名前がすでに存在しているものは許されません．この場合はエラーとなり名前は変更されません．

```
files 2
backup.n88 2    format.n88 2    setinf.n88 1    xfiles.n88 1    Quartz.     1
Cpaint.    1    DEMO   .   1    DEMOUT.    6    text01 asc 1    sample*bin  1

Ok
name "2:text01.asc" as "2:text"
Ok
files 2
backup.n88 2    format.n88 2    stinf.n88 1     xfiles.n88 1    Quartz.     1
Cpaint.    1    DEMO   .   1    DEMOUT.    6    text       1    sample*bin  1

Ok
■
```

5.5.7　3つの属性とは (SET)

SET は，指定したドライブ，ファイル，ファイル番号に属性を付ける働きをします．属性は属性文字で指定し，書き込み禁止（ライトプロテクト）属性は P，書き込み確認（リードアフタライト）属性は R で示します．属性の指定は次のように行います．

① SET　ドライブ番号，〝属性文字〞

② SET　ファイルディスクリプタ，〝属性文字〞

③ SET #ファイル番号，〝属性文字〞

①は，指定されたドライブに入っているディスケットに対して，属性が付けられます．②は，指定されたファイルに対してだけ，属性が付けられます．他のファイルには影響はありません．③はファイル番号が指定された場合で，それ以後の，そのファイルへの出力に対して属性が作用します．

属性の指定は，属性文字の P または R（大文字）で行います．これ以外の文字を指定すると，それまでの属性は解除されます．この属性は SET のみによって変更することが可能です．

書き込み禁止属性は，そのファイル（ディスケット）に対する PRINT #，PUT，WRITE #などの書き込み動作を全て禁止します．また，ファイルの KILL の実行もできなくなります．

書き込み確認属性は，書き込みを行った直後に，即そのデータを読み出しこの 2 つのデータを比較することによって，正しく書き込みが行われたかの確認を行わせるものです．

```
set "2:text","P"
Ok
kill "2:text"
File write protected
Ok
open "2:text" for output as#1
File write protected
Ok
■
```

5.5.8　プログラムの融合 (MERGE)

MERGE は，ファイルディスクリプタで指定されたプログラムファイルを現在メモリ上にあるプログラムといっしょにして 1 つのプログラムとします．

MERGE　ファイルディスクリプタ

指定されるプログラムファイルは，アスキー形式のファイルでなければなりません．またメモリ上のプログラムとファイルの中のプログラムに同じ行番号があった場合は，ファイルの中のその行が，メモリ上の行と置き換わります．

MERGEは，いくつかのプログラムをまとめて1つにする時に用います．プログラムを作る上で頻繁に使用するようなサブルーチンなどは，ライブラリとしてプログラムファイルを作っておき，必要な時にMERGEによってメインプログラムといっしょにするのです． N_{88}-BASICではサブルーチンにラベルを使用することができますから，このプログラムのライブラリとしての機能を十分に生かすことができます．

```
new
Ok
1000 *sub
1010 return
save "2:sub",a
Ok
new
Ok
100 gosub *sub
110 end
merge "2:sub"
Ok
list
100 GOSUB *SUB
110 END
1000 *SUB
1010 RETURN
Ok
■
```

5.5.9　大きなプログラムもこれで安心（CHAIN）

CHAINは，ファイルディスクリプタで指定されたプログラムをロードし，実行する働きをします．プログラムのチェイン（RUNの項参照）を行うためのものです．

CHAIN〔MERGE〕，ファイルディスクリプタ〔，行番号〕〔，　DELETE　行番号の範囲〕

CHAIN命令には次のような働きがあります．

① ファイルディスクリプタで指定されるプログラムをロードして実行します．

② 現在既に開かれているファイルはクローズしません．

③ 行番号はロードしたプログラムの実行開始行を指定します．この行番号が省略された時は，プログラムの最初から実行します．

④ プログラムをロードする前の変数をそのまま使うことができます．それにはALLオプションを指定します．

これを指定すると，全ての変数，配例の内容が引き渡されます．省略した場合は，いっさい引き渡されません．

また，必要な変数，配列のみを引き継ぎたい場合は COMMON 文を使います．

⑤ MERGE オプションは，現在メモリ上にあるプログラムと指定されたプログラムを MERGE して，指定された行番号から実行させるためのものです．この MERGE オプションを指定した場合は，指定されたプログラムファイルはアスキー形式のものでなければなりません．

⑥ DELETE オプションは，MERGE オプションが指定された場合のみに意味を持つもので，プログラムのマージを行う前に，メモリ上のプログラムの指定した行番号の範囲を削除します．

プログラムのチェインを行うための手順は，次のようなものです．

① プログラムが大きくなってメモリに入り切らない場合，そのプログラムをいくつかのモジュールに分けます．

② 各モジュールをどのようにチェインして実行するかを決めて，CHAIN 文を使ってプログラムを書く．

③ 各モジュールを別のファイル名でディスクにセーブしておく．アスキーセーブすることを忘れないように！

④ 実際にプログラムのチェインを行って，実行できるプログラムにする．

プログラムのチェインは，たくさんのモジュールプログラムを組み合せて，メモリにとうてい入らないような大きなプログラムを作る場合に使うと，有効な手段です．

```
new
Ok
1000 'chained program
1010 print "** chained **"
1020 print a%,a$
1030 end
save "2:test",a
Ok
new
Ok
100 'main program
110 print "** main **"
120 a%=1234 : a$="abcdef"
130 '
140 chain merge "2:test",1000,all
run
** main **
** chained **
 1234          abcdef
Ok
■
```

5.5.10 データの受け渡しには (COMMON)

COMMON は，CHAIN 文を使ってプログラムのチェインを行う時に，チェインされたプログラム間の変数の引き渡しを行います．

COMMON　変数名の並び

変数名の並びは，値を引き渡す変数の名前です．配列変数の場合は，配列変数名に〝()〟を付け加えて表します． COMMON 文には，いくつ変数名を並べても構いませんが，1つのプログラムの中で，COMMON 文に同じ変数名があってはいけません．

COMMON 文は， CHAIN 文が実行される前に実行されていなくては，その効力がありません．

なお全ての変数を引き渡したい場合は， CHAIN 文の ALL オプションを使う方が便利です．

```
new
Ok
100 'main program
110 print "** main **"
120 a%=1234 : a$="abcdef"
130 '
140 common a$
150 chain merge "2:test",1000
run
** main **
** chained **
 0             abcdef
Ok
■
```

5.5.11 属性とディスケットの未使用領域を知る（ATTR$, DSKF）

どちらも関数で，ファイル（ディスクファイル）に関する情報を与えます．

☆ ATTR$

ATTR$ は，フロッピーディスクあるいはファイルの属性を返します．この属性は SET 命令で付けられたものです．

① ATTR$ （ドライブ番号）

② ATTR$ （#ファイル番号）

③ ATTR$ （ファイル名）

①はフロッピーディスクに現在付けられている属性を，②はオープンされているファイルの現在の属性を返します．また③はそのファイルの属性を返します．

属性は全て文字列で求まります．

- 書き込み禁止　〝　P〟
- 書き込み確認　〝R　〟
- 属性なし　〝　〟

```
new
Ok
100 input a$
120 set "2:test",a$
130 print "/";attr$("2:test");"/"
140 print : goto 100
```

```
run
? P
/   P/

? R
/R  /

? p
/   /

?
Break in 100
Ok
set 2,"P"
Ok
save "2:test01"
File write protected
Ok
set 2," "
Ok
save "2:test01"
Ok
■
```

☆ DSKF

DSKFは，指定されたフロッピーディスクに関する情報を与える関数です．使用法はいくつかありますが，ここではその中でよく使われるものについて説明します．

① ディスクの未使用領域の大きさを求める．

DSKF（ドライブ番号）

ディスクの残りの容量をクラスタ単位で与えます．

② ディスクの構造についての情報を求める．

DSKF（ドライブ番号，機能）

機能をいろいろ指定することによって接続されているディスクの種類や構造を知ることができます．〈機能〉に与える数値は，0～11までであり，返される内容は次の通りです．

0――ディスクの最大のトラック番号を与えます．トラック番号は，0から始まりますから，そのディスクのトラックの総数は，それに＋1したものになります．ただし，以上は片面用ディスクの場合であり，両面用の場合，全体のトラック数は，それを2倍したものになります．

1――1トラックに何セクタあるかを返します．

2――そのディスクが片面用であるか，両面用であるかを与えます（片面＝0，両面＝1）．

3――トラック当りのクラスタ数を与えます．1クラスタはミニディスクの場合で2Kバイト，8インチの場合で約6.5Kバイトです．

以下，4～11まで指定することができますが，返す値はドライブの根本的な諸元で，DISK BASICが管理しているものです．これらの解説は，専門的にもなりますし，一般の使用には全く必要な

いと思われますので，省略します．もし詳しく知りたい方は，メーカー供給のユーザース・マニュアルを調べて下さい．

このDSKFにおいて，ディスクの未使用エリアの容量を知る以外の機能は，ディスクを使ったユーティリティープログラムの作成時に是非とも必要になるものです．たとえば，ミニ，8インチを全く意識せず，どちらでも使えるシステムなどです．メーカー供給のユーティリティープログラムの〝backupn88〟，〝formatn88〟などがよい参考になると思います．

```
100 print "drive No.    : 2"
110 print "track count :";dskf(2,0);"/drive"
120 print "sector count:";dskf(2,1);"/track"
130 print "side count  :";dskf(2,2)+1
140 print "free space  :";dskf(2);"[ｸﾗｽﾀ]"
150 end
run
drive No.    : 2
track count : 34 /drve
sector count: 16 /track
side count  : 1
free space  : 43 [ｸﾗｽﾀ]
Ok
■
```

5.5.12　機械語のプログラムには（BSAVE，BLOAD）

☆BSAVE

メモリ上の機械語プログラムを，ファイルディスクリプタで指定されるファイルにセーブします．

BSAVE　ファイルディスクリプタ，開始番地，長さ

メモリの開始番地から，長さのバイト数の内容を機械語プログラムとしてセーブします．

一般的な使い方としては，内蔵のモニタで作った機械語プログラムをディスクにセーブすることが考えられますが，その他，BASIC上の配列の内容をそっくりそのままデータファイルとする方法も考えられます．この場合，配列の最初の要素のアドレスと最後の要素のアドレスをVARPTR関数で調べ，セーブする開始アドレスと長さを求めます．この方法は，グラフィック画面からGET@命令で配列に格納したグラフィックパターンをファイルする方法として最も有効な手段です．

☆ BLOAD

ファイルディスクリプタで指定された機械語プログラムをメモリにロードします．

BLOAD　ファイルディスクリプタ〔，ロードアドレス〕〔，R〕

機械語がロードされる番地は，BSAVEで指定した開始番地と同じですが，ロードアドレスを指定すると，その番地からロードされます．BSAVEでセーブしたグラフィックパターンなどの

データをメモリにロードするのも，当然この命令です．

Rを指定すると，プログラムをロードした後，プログラムをロードした先頭番地より実行を始めます．

5.5.13　もう１つのWIDTH文（WIDTH）

☆ WIDTH

デバイスやバッファの文字数を設定します．

WIDTH　ファイルディスクリプタ，サイズ

ファイルディスクリプタで指定されたデバイスのバッファのサイズを設定します．指定できるデバイスは，プリンタと RS-232C ポートです．サイズは０～255の範囲であり，０は256とみなされます．

プリンタに対してこの命令を実行すると，プリンタの１行に印字される文字数は，サイズで指定したものとなります．

```
width "lpt1:",36
Ok
lfiles 2
Ok
■
```

WIDTH　ファイル番号，サイズ

ファイル番号で割り当てられているバッファ（チャネル）に対して，その大きさを設定します．

これを使ってコミュニケーションポートのバッファのサイズやプリンタの１行の表示文字数を自由に変えることもできます．

```
100 OPEN "lpt1:" FOR OUTPUT AS #1
110 PRINT #1,"1234567890123456789"
120 WIDTH #1,12
130 FOR I=1 TO 16
140  PRINT #1,"X";
150 NEXT I
160 CLOSE #1
170 END
```

なおそれぞれのバッファの大きさの初期値は，255に設定されています．

これら２つの命令は，一般の使用ではほとんど必要ないでしょう．

5.6　データファイルの作り方

これまでお話ししてきた命令のほとんどは，プログラムファイルを扱う命令でしたが，ここでは，データファイルの作り方について説明していきましょう．

5.6.1 データファイルとは

プログラム中で処理した結果のデータ（数値，文字等）は，どこかに保存しておかないと，電源OFFと同時に消えてしまいます．そこで一般的な手段としては，プリンタに印字する．カセットまたはディスクにファイルしておくなどの手段を使わなければなりません．

BASICの扱えるデータファイルには，シーケンシャルファイルとランダムファイル（ランダムアクセスファイル）の2種類があります．

シーケンシャルファイルとは，連続して書かれたデータの集まりで，前から順に書いた順序でしか読めなくなっています．データが1つの列をなして並んでいると考えることができます．

オーディオカセットは，1本のテープにデータを順に記憶する訳で，これはシーケンシャルファイルと呼ぶことができます．

シーケンシャルファイルのデータの構造を見てみましょう．

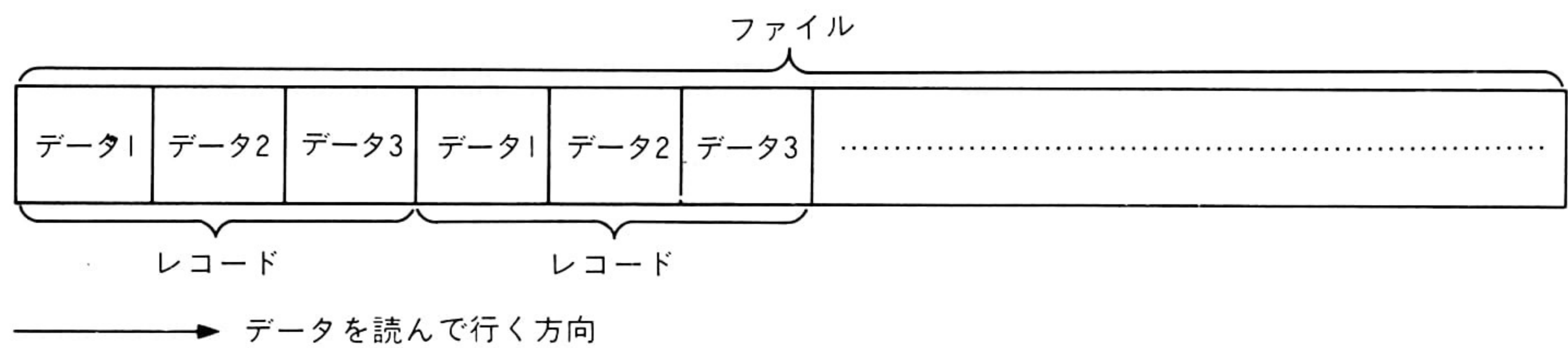



図6　シーケンシャルファイルのデータ構造

ファイルはたくさんのデータの集まりです．シーケンシャルファイルはこのデータを常に最初から一方向にしか読み進めません．レコードとは，一回に扱うデータの集まりです．この長さ（データの数）は自由に変えることができます．シーケンシャルファイルのレコード（データ）の追加は，常にファイルの最後に加えていきます．ファイルの中の一部のデータだけ書き変えるということはできません．

一方，ランダムファイルは，ファイルの中のデータを自由に何番目のものでも読むことができます．またファイルのデータの書き換えも自由に行うことができるのです．しかし，レコードの長さが固定であることや，ファイルの操作がシーケンシャルファイルに比べて，大分複雑であるという欠点もあります．

シーケンシャルファイル，ランダムファイルとも，それぞれ特徴のあるファイルですから，それぞれの特徴を理解して，処理の内容に合ったファイルを作らなければなりません．そのためには，第一にファイルの操作に慣れることです．

まず比較的操作の簡単なシーケンシャルファイルから扱ってみましょう．シーケンシャルファイルは，ランダムファイルに比べて見劣りするようですが，シーケンシャルファイルを使いこなせば，ほとんどのデータ処理が行えるのです．

5.6.2 シーケンシャルファイルの操作

ファイルの操作の手順は次のようになります．

① **ファイルの使用開始手続き**

② **入出力操作**

③ **ファイルの使用終了手続き**

データファイルを取り扱う場合は，常にこの手順で行います．それではどういう命令を使ってファイルを扱えばよいかをお話していきましょう．

☆OPEN

これはファイル操作を行うための，初めての使用開始手続きです．OPEN文では，ファイルディスクリプタによってどんなファイルを扱うかを指示し，そのファイルにファイル番号を付ける働きをします．

OPEN　ファイルディスクリプタ　FOR　モード　AS　ファイル番号

モードは，ファイルへのアクセスの方法を決めるものです．シーケンシャルファイルの場合には，次の3種類があります．

INPUT …………指定したファイルより入力を行うことを示します．
OUTPUT ………指定したファイルを作り，出力を行うことを示します．
APPEND ………すでに存在するファイルに，追加を行うことを示します．

INPUTとAPPENDモードでは，指定されたファイルは存在していなければなりません．逆にOUTPUTでは，存在しているファイルがあると，そのファイルの上から書き直しますから，古いファイルの内容は失われます．

ファイル番号は，1から15までの数字を使うことができますが，BASICの起動時に指定した数を越えてはいけません．また＃の記号は省略することもできます．

起動時の"How many files?"のメッセージを思い出して下さい．これはプログラム中で同時に使えるファイルの数を幾つにするかと聞いていたのです．ここで入力した数より多くは，1度にファイルを使うことはできません．

OPEN文によってファイルにファイル番号を対応づけたら，今後は入出力の操作をこの番号によって行うことができます．

OPEN文の例をいくつか示しておきましょう．

	ファイル	モード	ファイル番号
OPEN "1 : TEST" FOR OUTPUT AS #2	「TEST」	出力	2

（以後，フロッピーディスクドライブ上の"TEST"というファイルとの入出力は，ファイル番号によって行うことができます.）

OPEN　"SCRN:" FOR OUTPUT AS　#2　スクリーン　出力　2
OPEN　"KYBD:" FOR INPUT AS　#1　キーボード　入力　1

☆PRINT#／PRINT USING#

シーケンシャルファイルに，データを書き込む働きをします.

PRINT#ファイル番号，式の並び

ファイル番号を指定されるファイルに，式で表されるデータを書き込みます．ファイル番号で指定するファイルは，出力モードで開かれていなければなりません.

PRINT#文は，PRINT 文で画面へ出力するものと全く同じ形のものを出力します．ですからデータを効率よく出力し，後で正しく入力できるようにするにはちょっと工夫が必要です．そのためファイルへの出力をうまく行うために，WRITE#文が用意されています.

PRINT# USING 文も，ファイルへの出力を行うことを除いて，PRINT USING 文と全く同じです.

☆WRITE#

指定されたファイルにデータを書き込みます.

WRITE#ファイル番号，式の並び

WRITE#は，PRINT#とほとんど同じ働きをしますが，式の値の出力方法が少し違っています．出力にあたっては，不要な空白は除かれて，それぞれの値の間はコンマで区切られます．また文字列は引用符で括られて表示されます.

WRITE#は，区切り記号としてコンマを必ず出力し，不要な空白は除いて出力しますから，PRINT#と比べて，ファイルの使用領域を最小限にすることができます．シーケンシャルファイルのデータ出力には，格好の命令です．但し，この命令は DISK BASIC でのみ供給されています.

☆INPUT#／LINE INPUT#

指定されたファイルから，データを読み込む働きをします.

INPUT#ファイル番号，変数の並び

ファイル番号で指定されるファイルは，入力モードで開かれていなければなりません．ファイルから変数の数だけデータを読み込みます.

PRINT#による入力は，ファイルから入力が行われることを除いて，INPUT の入力と同じです．ですからファイル上のデータは，正しく INPUT#で入力ができるような形になっていなければなりません．

例えば，INPUT 文は文字列の入力を行う場合，余分な空白は無視する働きがあり，正しく入力するには引用符で囲まれなければなりません．INPUT#で入力を行う場合も，正しく入力を行わせるには，ファイル上の文字列を引用符で囲む必要が生じる場合もあるのです．例えば，文字列の前や後に意味のある空白を並べる場合等（"Computer　　"）はファイルへの出力をPRINT#よりも，WRITE#で行っていると便利なのです．

LINE INPUT#もファイルから入力を行うことを除けば，通常の LINE INPUT と同じく，キャリッジリターン・コードが来るまでの文字列を全て文字変数に読み込みます．例えば，コンマ（,）やダブルクォーテーション（"）などもすべて無視して読みこみますから，文章などを扱う場合，欠かすことが出来ません．

あまり必要はないでしょうが，アスキーセーブされているプログラムファイルであれば，このLINE INPUT#文で読み込むことができます．

☆INPUT $

INPUT$ は，指定されたファイルにより文字列を入力する関数です．

INPUT$（文字数，#ファイル番号）

ファイル番号で指定されるファイルより，文字数の長さの文字列を読み込みます．指定されたファイルは，入力モードで開かれたファイルでなければなりません．指定できる文字数は 1 ～255 までの範囲です．

☆EOF

EOF は指定されたファイルのデータが，終わりに達したかどうかを調べる関数です．

EOF（ファイル番号）

ファイル番号で指定したファイルのデータが，空になったなら真(－1)，そうでなくまだ残っているなら偽 (0) を値として返します．ファイル番号は，入力モードで開かれたファイルでなければなりません．

☆CLOSE

CLOSE は，ファイルの使用終了の手続きです．OPEN 文がファイルを開くのに対して，CLOSE 文はファイルを閉じる働きをします．

CLOSE ファイル番号の並び

指定されたファイル番号を閉じて,その番号を他のファイルが使うことができるようにします.ファイル番号を指定しなかった場合は,現在開いているファイルを全て閉じます.

ファイルを閉じると,もうそのファイルは入出力を行うことはできません.またファイルに使われていたファイル番号は開放され,他のファイルを開くために使うことができるのです.使用が終ったファイルは,必ず閉じておかなければなりません.

なお END とNEW を実行すると,全てのファイルは自動的に閉じられますから,プログラムの終了時に,END 文を書く習慣さえ付けておけば,ファイルを開いたまま終了することはなくなります.

例　CLOSE #2, 3　　　ファイル番号 2 と 3 のファイルを閉じる
　　CLOSE　　　　　　全てのファイルを閉じる

それではこれらの命令を使って,実際にどうやってファイル操作を行うのかをお話ししましょう.前に述べたようにファイル(シーケンシャルファイル)の操作には基本的な形があります.それは次のような手順です.

① OPEN 文によってファイルをオープンする.

② WRITE#, PRINT#などを使ってデータをファイルに書き込む.またはINPUT#, LINE INPUT#などを使ってデータをファイルから読み込む.

③ 入出力操作が終ったら,CLOSE 文を用いてファイルをクローズする.

5.6.3 シーケンシャルファイルの作成

それでは,ディスクファイルを使ってデータファイルを作ってみましょう.まず決めることは,1つのレコードのデータの形式をどうするかです.レコードは1度にファイルに入出力されるデータの組です.これは,幾つかのデータを集めて1つにして取り扱うのです.例えば,生徒の成績のファイルならば,生徒の名前と点数が1つのレコードとなるでしょうし,住所録のファイルならば,姓名,住所,電話番号などが1つのレコードとなるでしょう.

ここでは例として,レコードの形式が自由に変えられるようにしてみました.データとして扱う値は数値と文字列がありますが,このデータの数を指定できるようにした訳です.このレコードの形成を図に示します.

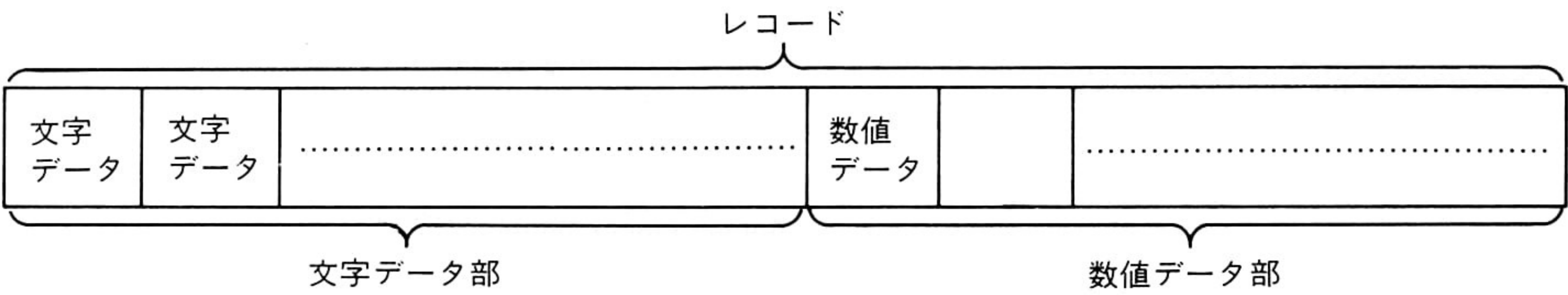



図7　レコード形式の例

文字データと数値データの数は，プログラムの実行時に決められるようにします．シーケンシャルファイルはレコードの長さを自由に変えることができますから，このようなことも簡単に行えるのです．

それでは実際のプログラムを示します．

```
100 'make sequential data file
110 INPUT "Character data count:";M%
120 INPUT "Number data count:";N%
130 DIM D$(M%),D(N%)
140 OPEN "TEST.01" FOR OUTPUT AS #1
150 '
160 *DATAINP
170 PRINT "Character data input"
180 INPUT " 1 :";D$(1)
190 IF D$(1)="end" THEN *EXIT
200 FOR I=2 TO M%
210  PRINT I;":"; : INPUT D$(I)
220 NEXT I
230 '
240 PRINT "Number data input"
250 FOR I=1 TO N%
260  PRINT I;":"; : INPUT D(I)
270 NEXT I
280 '
290 FOR I=1 TO M%
300  WRITE #1,D$(I)
310 NEXT I
320 '
330 FOR I=1 TO N%
340  WRITE #1,D(I)
350 NEXT I
360 GOTO *DATAINP
370 '
380 *EXIT : CLOSE #1 : END
```

出力ファイルとして〝TEST. 01〟というファイルをオープンしています．そして文字，数値のデータの数ほど配列を用意して，その配列にデータを読み込んで，後でまとめてファイルに書き込んでいます．1番目のデータとしてendが入力されるまで，このデータを読み込んでファイルに書き込む動作を続けます．

シーケンシャルファイルにデータを書き込む場合は，PRINT#文よりもWRITE#文の方が便利です．例えば，文字データを並べて書き込む時は，PRINT#文では次のようにしなければなりません．

```
PRINT#1, CHRS(34) ; D$ ; CHR$(34) ; CHR$(34) ; C$ ; CHR$(34)
```

PRINT#文はPRINT文と同じように，文字列をセミコロンで区切って出力させると，文字列がつながってしまうのです．これではこのファイルを読み込む時に，文字列がつながって読み込まれ，正しくデータを読むことができなくなるのです．それを防ぐためには，文字列を引用符やコンマで区切って出力しなければならない訳で，PRINT#では上の例のようになる訳です．ところがWRITE#文では，文字列の出力の際には引用符が付けられますから，このような心配をしなくて済むのです．

数値データや，文字データが続かないような場合は，PRINT#文でも構いません．しかし，WRITE

#文を使えば，不要な空白（符号や数値の後の空白）は出力せず，データをコンマで区切って出力しますから，PRINT#文よりファイルの領域が節約できます．

全てのデータの書き込みが終れば，CLOSE 文によってファイルを閉じて作業を終了します．

ではプログラムを実行させて，データをファイルに書き込んでみましょう．レコードの中の文字データの数は 1，数値データの数は 2 とします．end を入力して実行が終わったならば，FILES を実行してみて下さい．"TEST. 01" のファイルが作成されていることが分かるでしょう．

```
run
Character data count:? 1
Number data count:? 2
Character data input
 1 :? data-1
Number data input
 1 :? 1
 2 :? 2
Character data input
 1 :? data-2
Number data input
 1 :? 10
 2 :? 20
Character data input
 1 :? data-3
Number data input
 1 :? 100
 2 :? 200
Character data input
 1 :? end
Ok
files
backup.n88 2    format.n88 2    seting.n88 1    xfiles.n88 1    Quartz.     1
Cpaint.     1   DEMO   .    3   DEMOUT.     6   TEST    01
Ok
■
```

5.6.4 シーケンシャルファイルからのデータの読み出し

今度は，ファイルよりデータを読み出してみましょう．それには，ファイルを INPUT モードでオープンしなければなりません．また，データを読み出す場合には，そのファイルのレコードの形式がどうなっているかに注意しなければなりません．ファイルに書き込まれているデータと，それを読み込む変数の型は同じでなくてはならないのです．

それでは，先程の PROGRAM1 で作成したデータファイルを読み込むプログラムを示します．

```
100 'Read data file
110 OPEN "TEST.01" FOR INPUT AS #1
120 M%=1 : N%=2
130 '
140 *DATAINP
150 FOR I=1 TO M%
160  INPUT #1,D$
170  PRINT D$;":";
180 NEXT I
190 '
200 FOR I=1 TO N%
210  INPUT #1,D
220  PRINT D;
230 NEXT I : PRINT
240 GOTO *DATAINP
```

先程の "TEST. 01" ファイルをオープンし，レコードの形式に従って，文字データをその数だ

け読み込みます．その動作をファイルのデータが無くなるまで繰り返す訳です．

```
run
data-1: 1  2
data-2: 10  20
data-3: 100  200
Input past end in 160
Ok
■
```

実行してみると分かりますが，このプログラムでは，正常に終了しません．ファイルのデータが無くなると，Input past end エラーが生じます．それにこのプログラムには，ファイルをクローズする処理がありません．このプログラムを正常に終了させるためには，ファイルが終りになったかどうかを，検出しなければならないのです．それには，EOF 関数を使います．EOF 関数はファイルが終ったかどうか調べる関数です．

プログラムを次のように変更して下さい．

```
list
100 'Read data file
110 OPEN "TEST.01" FOR INPUT AS #1
120 M%=1 : N%=2
130 '
140 *DATAINP
150 IF EOF(1) THEN *EXIT
160 FOR I=1 TO M%
170  INPUT #1,D$
180  PRINT D$;":";
190 NEXT I
200 '
210 FOR I=1 TO N%
220  INPUT #1,D
230  PRINT D;
240 NEXT I : PRINT
250 GOTO *DATAINP
260 '
270 *EXIT : CLOSE #1 : END
Ok
run
data-1: 1  2
data-2: 10  20
data-3: 100  200
Ok
■
```

今度は正常に終了しました．EOF(1) という関数は，ファイル番号 1 のファイルが終りに達すると値が真となるのです．ファイルが終れば，そのファイルをクローズして終了です．

ところで，ファイルの作成プログラムでレコードの形式を自由に決めることができるようにしたのですが，そのためいろいろなレコードの形式のファイルができてしまうと，このレコード形式ごとの読み込みプログラムを作らなければならなくなってしまいます．また，そのファイルがどのようなレコード形式なのかも憶えておかなければ，そのファイルが正しく読めないことにもなるのです．

この問題を解決するには，ファイルの中にそのレコード形式を示す情報を含めておけばよいのです．ファイルのデータの読み込みに先だって，まずその情報をファイルから読んで，どのよう

なレコード形式になっているかを知ればよい訳です．それにはファイルを作成する時に，その先頭にレコードの形成を表すデータを書いておきます．そしてファイルの読み込みの時にまず，そのデータを読めばよい訳です．

それを行ったプログラムを次に示します．このファイルではレコードの文字データと数値データの数を示す数値が，ファイルの先頭に2つ書いてあります．

```
list
100 'Make data file
110 INPUT "Character data count:";M%
120 INPUT "Number data count:";N%
130 DIM D$(M%),D(N%)
140 OPEN "TEST.01" FOR OUTPUT AS #1
150 WRITE #1,M%;N%
160 '
170 *DATAINP
180 PRINT "Character data input"
190 INPUT " 1 :";D$(1)
200 IF D$(1)="end" THEN *EXIT
210 FOR I=2 TO M%
220  PRINT I;":"; : INPUT D$(I)
230 NEXT I
240 '
250 PRINT "Number data input"
260 FOR I=1 TO N%
270  PRINT I;":"; : INPUT D(I)
280 NEXT I : PRINT
290 '
300 FOR I=1 TO M%
310  WRITE #1,D$(I)
320 NEXT I
330 '
340 FOR I=1 TO N%
350  WRITE #1,D(I)
360 NEXT I
370 GOTO *DATAINP
380 '
390 *EXIT : CLOSE #1 : END
Ok
run
Character data count:? 1
Number data count:? 2
Character data input
 1 :? data(1)
Number data input
 1 :? 1
 2 :? 2

Character data input
 1 :? data(2)
Number data input
 1 :? 3
 2 :? 4

Character data input
 1 :? data(3)
Number data input
 1 :? 5
 2 :? 6

Character data input
 1 :? end
Ok
■
```

```
list
100 'Read data file
110 OPEN "TEST.01" FOR INPUT AS #1
120 INPUT #1,M%,N%
130 '
140 *DATAINP
150 IF EOF(1) THEN GOTO *EXIT
160 FOR I=1 TO M%
170  INPUT #1,D$
180  PRINT D$;":";
190 NEXT I
200 '
210 FOR I=1 TO N%
220  INPUT #1,D
230  PRINT D;
240 NEXT I : PRINT
250 GOTO *DATAINP
260 '
270 *EXIT : CLOSE #1 : END
Ok
run
data(1): 1  2
data(2): 3  4
data(3): 5  6
Ok
■
```

5.6.5 シーケンシャルファイルへのデータの追加

今度は，シーケンシャルファイルへのデータの追加の仕方についてお話ししましょう．ファイルにデータを追加するためには，そのファイルを APPEND モードでオープンしなければなりません．そして，データをファイルに書き込んでいけば，新しいデータファイルは，ファイルの後ろから追加されていきます．

一般的なファイルに，データの追加を行うプログラムは，そのファイルを作成するプログラムの OPEN 文のモードを APPEND モードに変えるだけで済みます．ところが，ここで使っているファイルはレコード形式がさまざまであるため，まずファイルの先頭のレコード形式を示すデータを読まなければなりません．この点がこのファイルにデータを追加する上での注意すべきことです．

それでは実際のプログラムを示します．

```
100 'Append data
110 OPEN "TEST.01" FOR INPUT AS #1
120 INPUT #1,M%,N%
130 CLOSE #1
140 '
150 OPEN "TEST.01" FOR APPEND AS #1
160 '
170 *DATAINP
180 PRINT "Character data input"
190 INPUT " 1 :";D$(1)
200 IF D$(1)="end" THEN *EXIT
210 FOR I=2 TO M%
220  PRINT I;":"; : INPUT D$(I)
230 NEXT I
240 '
250 PRINT "Number data input"
260 FOR I=1 TO N%
270  PRINT I;":"; : INPUT D(I)
280 NEXT I
290 '
300 FOR I=1 TO M%
```

```
310  WRITE #1,D$(I)
320 NEXT I
330 '
340 FOR I=1 TO N%
350  WRITE #1,D(I)
360 NEXT I
370 GOTO *DATAINP
380 '
390 *EXIT : CLOSE #1 : END
```

まずファイルのレコード形式を調べるために入力モードでファイルをオープンします．そしてそれを調べた後にファイルをクローズし，今度はデータを追加するために APPEND モードでオープンしています．

以後このプログラムを実行するごとに，ファイルのデータは増えていきます．

```
run
Character data input
 1 :? data(4)
Number data input
 1 :? 7
 2 :? 8
Character data input
 1 :? data(5)
Number data input
 1 :? 9
 2 :? 10
Character data input
 1 :? end
Ok
■
```

5.6.6 シーケンシャルファイルのデータの修正

今度は，シーケンシャルファイルのデータの修正は，どうやって行えばよいかを考えてみましょう．シーケンシャルファイルは，そのままではファイルのデータを修正することはできません．ですから，修正した結果を，別の新しいファイルに書き込んでいく作業をしなければなりません．

次のプログラムは，ここで作ったファイルの修正を行うものです．

```
100 '
110 OPEN "TEST.01" FOR INPUT AS #1
120 OPEN "TEST.WRK" FOR OUTPUT AS #2
130 INPUT #1,M%,N%
140 WRITE #2,M%,N%
150 DIM D$(M%),D(N%)
160 '
170 *DREAD
180 IF EOF(1) THEN *EXIT
190 PRINT : PRINT "Character data"
200 FOR I=1 TO M%
210  INPUT #1,D$(I) : PRINT I;":";D$(I)
220 NEXT I
230 PRINT "Number data"
240 FOR I=1 TO N%
250  INPUT #1,D(I) : PRINT I;":";D(I)
260 NEXT I
270 IF WRTFLG=1 THEN *DWRITE
280 '
290 PRINT : PRINT "ﾃﾞｰﾀ ﾉ ﾍﾝｺｳｶﾞ ｱﾘﾏｽｶ "
300 INPUT "(1:yes, 2:no, 3:delete, 4:end)";C$
310 IF VAL(C$)<1 THEN 300
320 ON VAL(C$) GOTO *CHANGE,*DWRITE,*DREAD,*CEND
```

```
330 GOTO 300
340 '
350 *CHANGE
360 PRINT "Chracter data"
370 FOR I=1 TO M%
380  PRINT I;":"; : INPUT D$(I)
390 NEXT I
400 PRINT "Numvber data"
410 FOR I=1 TO N%
420  PRINT I;":"; : INPUT D(I)
430 NEXT I
440 GOTO *DWRITE
450 '
460 *CEND
470 WRTFLG=1
480 '
490 *DWRITE
500 FOR I=1 TO M% : WRITE #2,D$(I) : NEXT I
510 FOR I=1 TO N% : WRITE #2,D(I) : NEXT I
520 GOTO *DREAD
530 '
540 *EXIT
550 CLOSE #1,2 : KILL "TEST.01"
560 NAME "TEST.WRK" AS "TEST.01"
570 END
```

このプログラムでは，修正を行ったデータを格納しておくために，〝TEST. WRK〟というファイルを使っています．〝TEST. 01〟ファイルから1レコードずつデータを読み込み，修正が無ければそのレコードをそのまま〝TEST. WRK〟ファイルに書き込みます．もし修正を行うならば，各データについて修正を行うかどうかを尋ねて，必要ならば修正を行って〝TEST. WRK〟ファイルに書き込みます．削除の場合は，データの書き込みは行わず．次のレコードを読み込みます．そして修正が終ると，〝TEST. 01〟ファイルの残りのデータは全て〝TEST. WRK〟ファイルに追加されます．

終りに〝TEST. 01〟ファイルを抹消して，〝TEST. WRK〟の名前を〝TEST. 01〟に変更して終了します．

☆プログラム例

シーケンシャルファイル，チャンネルを活用したプログラムの例を示しましょう．このプログラムはアスキー形式でセーブされたプログラムファイルの内容を表示するものです．アスキー形式のプログラムファイルは，データファイルとして扱うことができるのです．内容の表示は，画面とプリンタが選択できるようになっています．

```
100 'Print data file
110 INPUT "Data file name:";NM$
120 '
130 *SELECTDEVICE
140 PRINT "Output dvice"
150 INPUT "(1:screen,2:printer)";Y$
160 IF Y$="1" THEN DV$="scrn:" : GOTO *OPENFILE
170 IF Y$="2" THEN DV$="lpt1:" ELSE *SELECTDEVICE
180 '
190 *OPENFILE
200 OPEN NM$ FOR INPUT AS #1
210 OPEN DV$ FOR OUTPUT AS #2
220 '
230 *FILEINP
```

```
240 IF EOF(1) THEN *EXIT
250 LINE INPUT #1,A$
260 PRINT #2,A$
270 GOTO *FILEINP
280 '
290 *EXIT : CLOSE #1 : END
```

このプログラムでは，最初に表示を行うファイルの名前と，画面とプリンタのどちらを出力にするかを尋ねてきます．画面か，プリンタの指定された方のデバイスをオープンして，後はそのファイルに対して出力を行うようになっています．こうすることによって，画面とプリンタに対して，それぞれの命令を使い分ける必要がなくなります．このように，ファイルの出力をどのデバイスにするかをOPEN文で指定するだけで，1つのPRINT#命令で，自由に出力デバイスを選んで出力が行えるのです．

プログラムファイルからの1行ごとの読み込みは，LINE INPUT#文を使っています．この命令は，文字列の中にコンマや引用符が含まれていても，文字列が区切られることはありません．ファイルの中の文字列のデータを，キャリッジリターン・コードまで全ての文字を読み込みます．

5.6.7 ランダムファイルの操作

ランダムファイルは，シーケンシャルファイルに比べて次のような特徴があります．

① ランダムアクセスができる．シーケンシャルファイルのようにファイルの前から順に読んでいくのではなく，何番目のレコードでも自由にアクセスできます．

② ファイル内のデータを自由に書き換えられます．つまりファイルからの読み書きが自由にできるのです．またデータの追加もファイルの最後に行えます．

③ レコードの大きさは，256バイトに固定されています．シーケンシャルファイルの場合は，自由に変えることができました．

④ シーケンシャルファイルに比べて，取り扱いが複雑です．

以上のような特徴があります．

それでは，ランダムファイルを扱うための命令について述べていきましょう．

☆OPEN

シーケンシャルファイルの時のような，モードの指定はありません．ランダムファイルはファイルのどこからでも読み書きできるのです．

OPEN 〝ファイルディスクリプタ〞AS ファイル番号

ファイルディスクリプタは，ランダムアクセスを行うファイルを指定します．指定したファイルが存在しない場合は，新しくファイルが作られます．

☆CLOSE

動作はシーケンシャルファイルの場合と同じです．使用が終ったファイルを閉じます．

☆PUT

ランダムファイルの書き込みを行います．書き込みは，レコードの単位で行われます．

PUT ファイル番号，レコード番号

ファイル番号は，ランダムアクセスを行うディスクファイルを指定します．レコード番号で指定されるレコードに，ランダムファイルバッファの内容が書き込まれます．レコード番号を省略した場合は，直前にアクセスが行われたレコードの次のレコードに書き込みます．

PUT ファイル番号，数

これはディスクファイル以外のファイルに対して書き込みを行います．数はバッファからファイルに出力する文字の数で，0～255の範囲で指定します．この数を省略するか，または0を指定すると256文字をファイルに出力します．

PUT 文でファイルに出力するデータは，FILED 文，LSET／RSET 文によって，あらかじめ用意されていなければなりません．

☆FIELD

ファイルバッファをランダムファイルバッファとして使えるようにします．そのためにファイルバッファの中に，文字変数の領域を割り当てます．1レコード256文字をどのように使うかを決めるものです．

FIELD ファイル番号，フィールド幅 AS 文字変数，…………

フィールド幅は，文字変数に割り当てる文字数です．256文字以内ならば，いくつ文字変数に割り当てても構いません．FIELD 文の例を次に示します．

FIELD #1, 2 AS A$, 4 AS B$, 8 AS C$

この例では256文字の先頭から，2文字を A$，4文字を B$, 8文字を C$ に割り当てています．残りは未使用で無駄になってしまいます．

FIELD 文で文字変数を割り当てることから分かるようにランダムファイルの読み書きは常に文字列で行われます．

☆LSET／RSET

これらは，ランダムファイルバッファに割り当てた文字変数の領域に，文字列を代入する働き

をします.

LSET 文字変数＝文字列
RSET 文字変数＝文字列

文字変数は，FIELD 文で定義したものでなければなりません．LSET 文は，文字列を領域に左詰めで，RSET 文は，右詰めで代入します．空いた部分には空白が入れられます．もし文字列の長さが指定した変数の領域を越えた場合は，両方とも右側の部分が無視されます．

LSET B$＝〝PC〞
RSET C$＝〝－8801〞

ランダムファイルバッファの文字変数の代入には，必ず LSET／RSET 文を使います．普通の代入や，INPUT 文は使用してはいけません！

☆GET
ランダムファイルより，データ（1レコード）の読み込みを行います．指定されたレコードよりデータを読んで，ランダムファイルバッファに移します．

GET ファイル番号，レコード番号

ファイル番号は，ランダムアクセスを行うディスクファイルを指定します．レコード番号で指定されたレコードのデータをファイルバッファに読み込みます．レコード番号が省略された場合は，直前にアクセスされたレコードの次のレコードを読み込みます．

GET ファイル番号，数

ディスクファイル以外のファイルから，指定された数だけ文字をバッファに読み込みます．数は0～255の範囲で，省略するか，0を指定すると256文字だけ読み込みます．
GET 文で読み込んだデータは，FIELD 文で定義された文字変数に割り当てられ，それらの変数によってバッファからデータを取り出すことができます．

☆LOF
ファイル番号で指定されるランダムファイルの最大のレコード番号を与える関数です．

LOF（ファイル番号）

GET 文でファイルのデータを読む時は，この数値より大きいレコード番号は指定できません．またランダムファイルにデータの追加を行う時は，この値より1つ大きいレコード番号を指定して PUT 文を実行します．

このLOF関数の値は，ファイルのレコードの数が増えるごとに新しくなります.

☆LOC

ファイル番号で指定されるランダムファイルにおいて，そのファイルに対して，最後に読み書きされたレコード番号を与える関数です.

LOC（ファイル番号）

ファイルに対して GET／PUT 命令を実行すると，LOC 関数の値は更新されます.

また LOC 関数は，シーケンシャルファイルに対しては，ファイルがオープンされてから，これまで読み書きされたレコードの番号(256バイトごとに区切って番号を付けたもの)を与えます.

☆MKI$／MKS$／MKD$

ランダムファイルの書き込みは，文字列で行わなければなりません．そのため，数値データをランダムファイルに書き込みたい場合は，一度文字型に変換する必要があるのです．MKI$／MKS$／MKD$命令は，それぞれ整数型，単精度型，倍精度型の数値を文字型に変換する働きをします.

ところで，数値を文字に変換する関数には，STR$ 関数がありました．しかし，この関数は数値を PRINT 文で表示した場合と同じ形式に変換するものでした．これでは数値の大きさが変ると文字列の長さも変る場合があり，FIELD 文で変数を割り当てる時に，無駄が生じます.

MKI$／MKS$／MKD$ 関数はそれぞれの型の数値を，常に 2 文字，4 文字，8 文字の文字列に変換します．これは整数型，単精度型，倍精度型の数値の内部表現が，2 バイト，4 バイト，8 バイトで表されており，これらの関数は内部表現の 1 バイトを 1 文字に変換する働きをするのです.

☆CVI／CVS／CVD

これら 3 つの関数は，ランダムファイルから読み込んだデータを，文字列からもとの数値に戻す働きをします．CVI／CVS／CVD の関数は，それぞれ MKI$／MKS$／MKD$ の関数の逆関数となっているのです.

これらの関数を使うときには，同じ型を変換する関数を対にして用いなければなりません．MKS$関数で変換された文字は，必ず CVS 関数で数値に戻さなければならないのです.

また，FIELD 文で文字数の大きさを割り当てる場合には，各数値の型に必要なバイトを考えて，必ず必要な数の文字数を割り当てるように注意して下さい.

これまでに説明してきた命令の，ランダムファイルの操作における役割りを図にしてみました.

ランダムファイル
先頭レコード
文字型データ
ランダムファイル
バッファ(256文字)
文字データ
LSET
RSET
GET
PUT
レコード
(256文字)
MKI$/MKS$/MKD$
数値データ
CVI/CVS/CVD
文字変数に
割り当てた領域
FIELD
LOF
最終レコード
OPEN
CLOSE

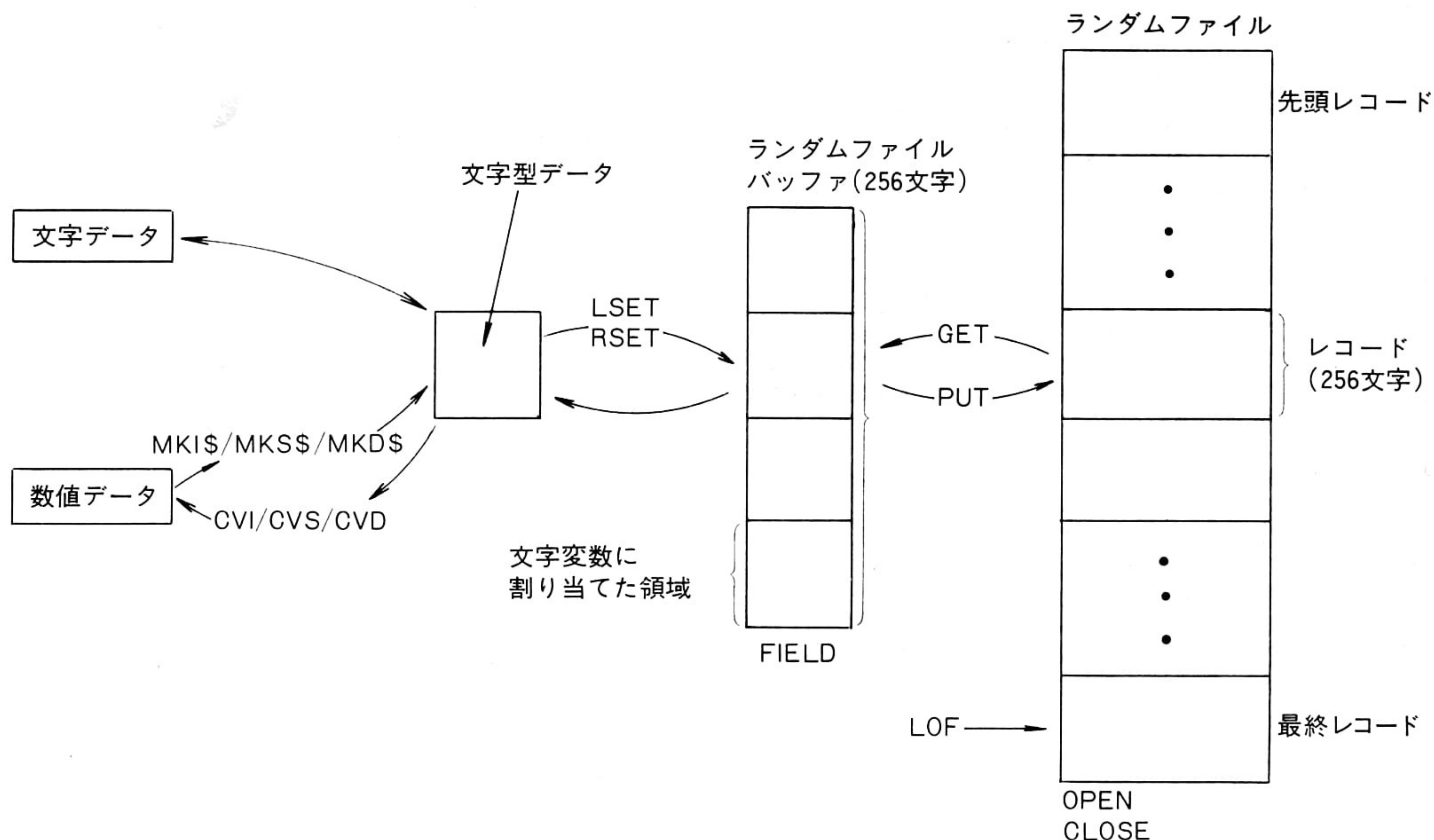



図8　ランダムファイルの操作

5.6.8 ランダムファイルの作成

ランダムファイルを操作する手順は，次の様になります．

① ランダムファイルをオープンします（OPEN 文）

② ランダムファイルバッファに変数を割り当てます（FIELD 文）

③ ファイルへの書き込みの場合は，LSET／RSET 文を使ってバッファにデータをセットします．そして PUT 文によってファイルに書き込みます．

④ ファイルからの読み込みの場合は，GET 文によってデータをバッファに読み込みます．そして，バッファに割り当てられた文字変数よりデータを取り出します．

③，④の作業が終了したなら，ファイルを閉じます．

③の作業において数値データを扱う場合は，MKI$／MKS$／MKD$ 関数を使って，数値を文字に変換し，CVI／CVS／CVD 関数を使って，文字を数値に戻します．

```
100 '
110 INPUT "File name ";FLN$
120 OPEN FLN$ AS #1
130 FIELD #1,13 AS ID$,4 AS D1$,4 AS D2$
140 '
150 *MENU
160 PRINT
170 PRINT "1:ﾃﾞｰﾀ ﾆｭｳﾘｮｸ, 2:ﾃﾞｰﾀ ｹﾝｻｸ, 3:ﾃﾞｰﾀ ﾍﾝｺｳ"
180 PRINT "4:ｽﾍﾞﾃﾉ ﾃﾞｰﾀ ﾋｮｳｼﾞ, 5:ｵｼﾏｲ"
190 INPUT "ﾄﾞﾚﾆｼﾏｽｶ ";MN : PRINT
200 ON MN GOTO *DATAINPUT,*DATASEARCH,*DATACHANGE,*DATAPRINT,*WORKEND
210 GOTO *MENU
220 '
```

```
230 *DATAINPUT
240 INPUT "ID name";D$
250 INPUT "Number data1";D1
260 INPUT "Number data2";D2
270 LSET ID$=D$ : RSET D1$=MKS$(D1)  : RSET D2$=MKS$(D2)
280 PUT #1,LOF(1)+1
290 GOTO *MENU
300 '
310 *DATASEARCH
320 GOSUB *IDSEARCH : PRINT
330 IF FINDFLG<>1 THEN *MENU
340 PRINT "ID name      :";ID$
350 PRINT "Number data1:";CVS(D1$)
360 PRINT "Number data2:";CVS(D2$)
370 GOTO *MENU
380 '
390 *DATACHANGE
400 GOSUB *IDSEARCH
410 IF FINDFLG<>1 THEN *MENU
420 INPUT "Number data1:";D1
430 INPUT "Number data2:";D2
440 IF D1<>0 THEN RSET D1$=MKS$(D1)
450 IF D2<>0 THEN RSET D2$=MKS$(D2)
460 PUT #1,RN : GOTO *MENU
470 '
480 *DATAPRINT
490 PRINT "ID name          Data1          Data2"
500 FOR I=1 TO LOF(1)
510  GET #1,I
520  PRINT ID$,CVS(D1$),CVS(D2$)
530 NEXT : GOTO *MENU
540 '
550 *WORKEND
560 CLOSE #1 : END
570 '
580 *IDSEARCH
590 INPUT "ID name ";D$
600 D$=LEFT$(D$+SPACE$(13),13)
610 RN=0 : RNMAX=LOF(1) : FINDFLG=0
620 WHILE FINDFLG=0 AND RN<RNMAX
630  RN=RN+1 : GET #1,RN
640  IF ID$=D$ THEN FINDFLG=1
650 WEND
660 IF FINDFLG=0 THEN PRINT "Not found!"
670 RETURN
```

6章　割り込みとその使い方

あなたは「割り込み」とか「インタラプト（Interupt）」という言葉を聞いた事がありますか？始めて聞く人も多いかもしれませんが，この割り込みというのは計算機にとって非常に重要な機能で，現在の計算機でこの機能が無い事は絶対にありません．ではこの重要な割り込み機能を知らない人が多いのはなぜでしょう．それはこの機能が重要で欠く事が出来ないけれど，割り込みを使っていると意識していなかったりします．つまり割り込みは，無いと困るが使わなくても結構やっていける機能なのです．この章ではこの割り込みについて，その仕組みや意義，そしてBASICでの使い方について説明していきます．

6.1　割り込みって何でしょう

6.1.1　割り込み処理の流れ

「割り込みだって!?　何だそりゃ，使った事も聞いた事も無いぞ」なんて言わないで下さい．あなたは，もう十分に使っているのですよ．ただそれと気付かないだけで．キーボードの上の隅を見て下さい．STOP というキーがありますね．このキーを押すとどうなるのでしょう．次のプログラムを走らせて下さい．

```
100 PRINT "ウコ゛イテ マッセ"
110 GOTO 100
```

たった2行の簡単なプログラムで，100行110行と繰り返し実行する，単なる無限ループのプログラムです．実行を止めるにはSTOPキーかCTRL-C（コントロールキーを押しながらCのキーを押す）のお世話になります．

```
Ok
100 print "ウゴイテ マッセ"
110 goto 100
run
ウゴイテ マッセ
ウゴイテ マッセ
ウゴイテ マッセ
ウゴイテ マッセ
^C
Break in 100
Ok
■
```

またプログラムのリストを見るにはLISTと打ちますが，その中断にはやはりSTOPキーを押します．そしてSTOPキーを押すとコマンド待ちの状態に戻りますが，実はこれが割り込みなのです．

割り込みとは内部あるいは外部のある要因，例えばキーが押されたとか，ライトペンが押されたとか，この場合はSTOPキーが押された事により，指定された処理を行う事なのです．そしてこの要因はいつ起こるか分らないので，常にソフトウェアによって監視している訳にもいかず，そこでハードウェアによって強制的にその時だけ，指定された割り込み処理を行う様になっているのです．

では，次のプログラムはどうでしょう．

```
100 IF INKEY$="" THEN 100
110 PRINT "オシマイ"
```

走らせてみて下さい．何か適当なキー（STOPキー以外）を押すと実行が終りますね．確かにこのプログラムでは外部からの入力によって実行が終りますが，これは割り込みではありません．100行でINKEY$によりキーが押されたかどうか常に調べています．つまり外部からの入力を，ソフトウェアにより常に監視している訳で，割り込みとは呼ばないのです．実はこの違いが割り込みで最も重要な点なのです．

割り込みでは，ハードウェアによって指定された処理を起動すると述べましたが，これはソフトウェアで調べている必要が無いという長所を示しているのです．ですから，その分の時間は別の事をすることが出来ます．その代りに割り込み処理では次の事が必要となります．

① **割り込みが起った時の処理の宣言　(ON〜GOSUB 文)**

② **割り込みを許可する　(〜 ON 文)**

③ **必要な時には割り込み機能を停止する　(〜 STOP 文　または　〜 OFF 文)**

④ **割り込み処理からの復帰　(RETURN 文)**

次の図を見て下さい．

メインプログラム
on～GOSUB文 …………割り込み時に始める行番号の指定
～ON文 …………割り込みを許可する
割り込み
～STOP文
割り込み　割り込みを保持
～ON文
～OFF文
割り込み　無視
割り込み処理プログラム
Return文

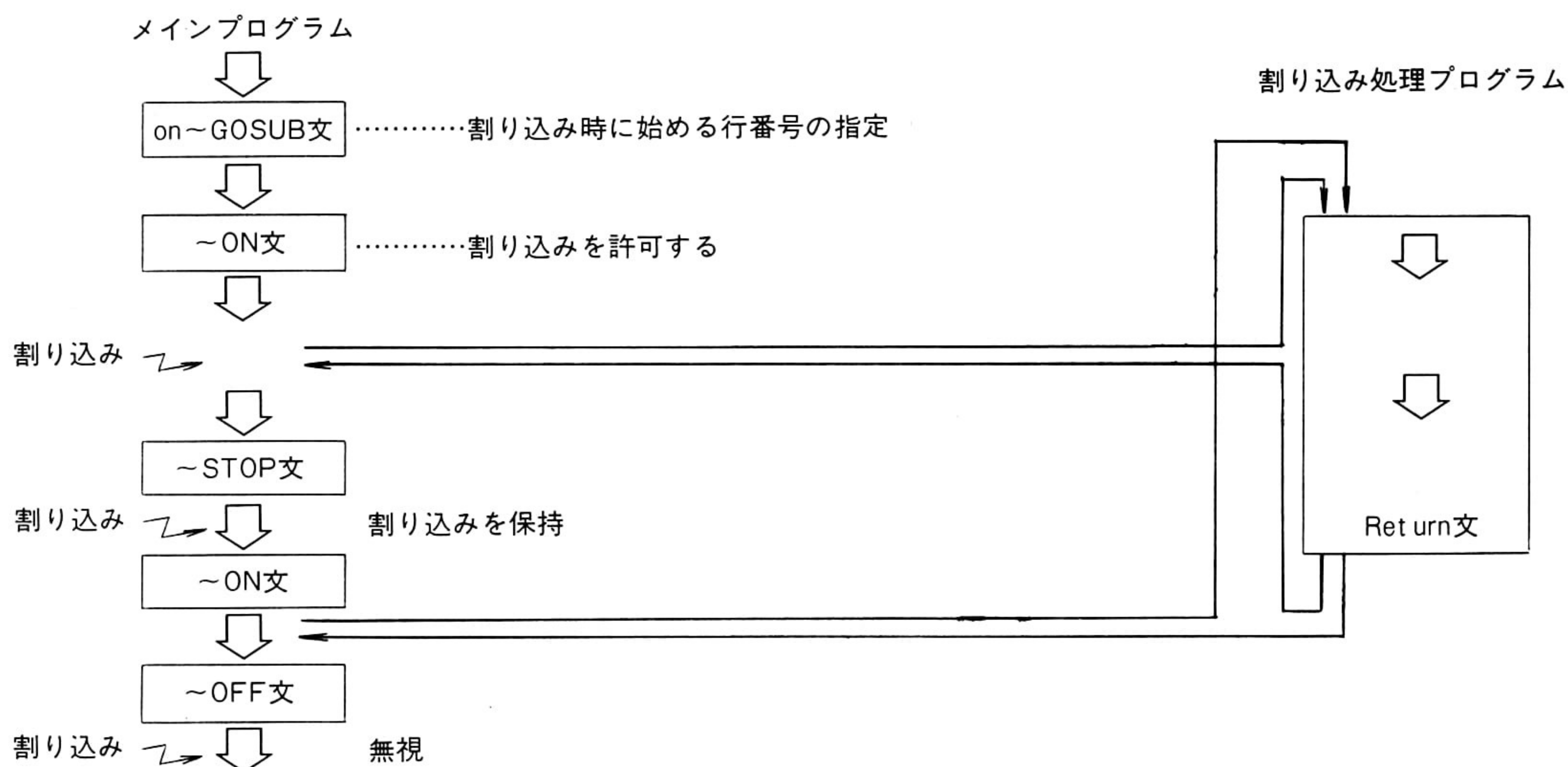



図1　割り込みを使ったプログラム

6.1.2 BASICで割り込みを使うには

まず最初に,割り込みが起こったら何をするかを指定しなければなりません.そこでON～文により，その時に始めるプログラムの行番号を与えておきます．ここまでは，まだ割り込み禁止状態にあるので，次に～ON文によって許可します．これで割り込みが受け付けられる様になりました．メインプログラムで何をしていても，割り込みが起これば，即座に指定された所から実行を始めます．そして，そのまま実行を続けたら……ハテ途中でやめたメインルーチンはどうなるのでしょう．困りますね．そこで，サブルーチンでお馴染みのRETURN文の登場となる訳です．RETURN文を実行すると，再びメインルーチンへと戻ってくれます．まるでサブルーチンと同じですね．

メインルーチン
GOSUB文
GOSUB文
サブルーチン
RETURN文

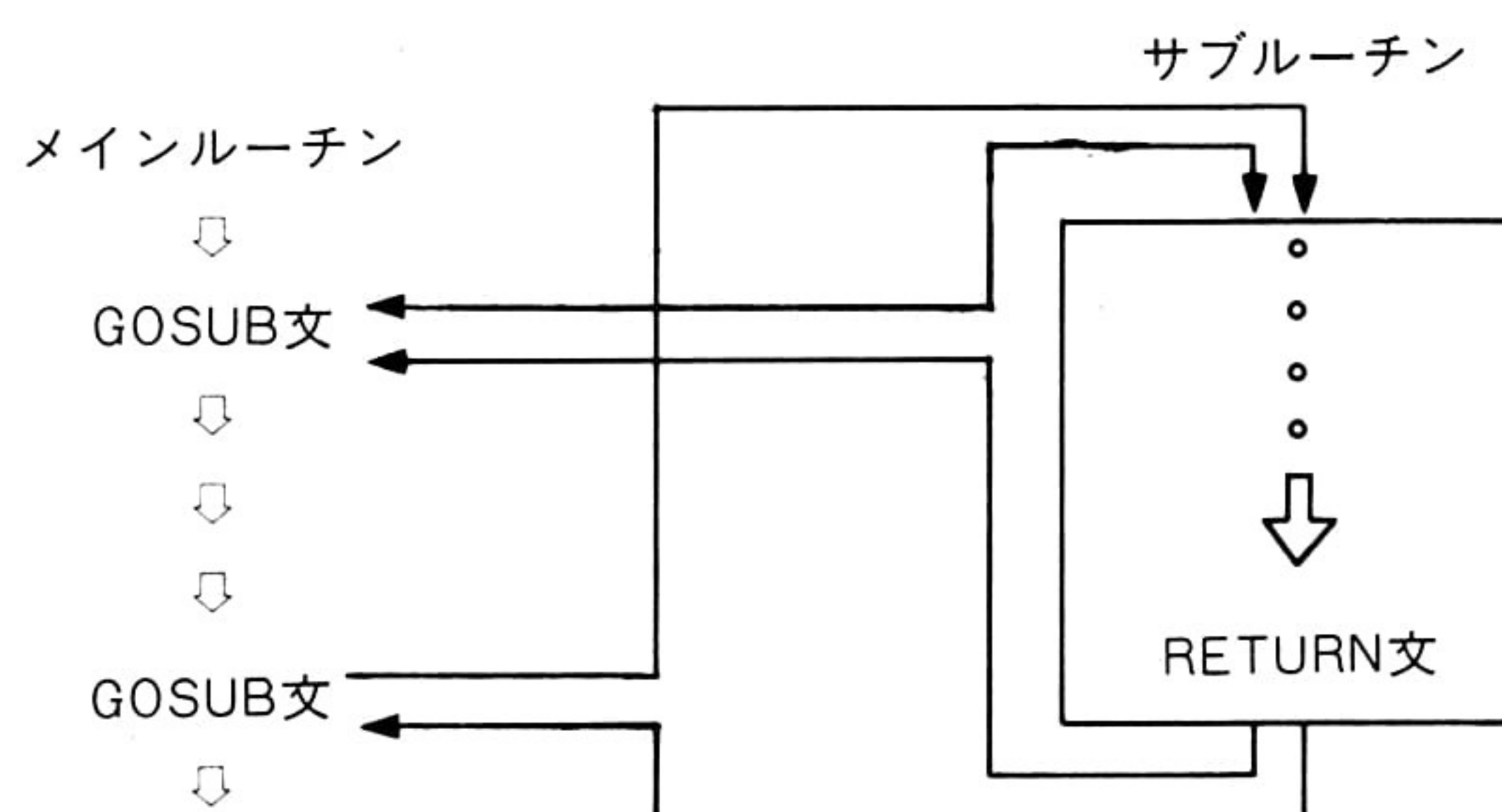



図2　サブルーチンの呼び出し

そう実は割り込み処理とは，特殊なサブルーチンとも言えるのです．

一般のサブルーチンではGOSUB文によって呼び出されRETURN文により戻りますが,GOSUB文を割り込みに替えれば，割り込み処理となるのです．

さて，割り込み処理ルーチンから戻ると，再びメインルーチンを続けますが，なりふり構わず割り込まれては困るのです．「ここを実行している時は割り込みを待たせておきたい！」という場合があります．そういう場合には～STOP文を使います．この文を実行すると，以後，割り込みが起こっても割り込み処理は行いません．ただ割り込みがあった事を覚えておき，～ON文により再び許可された後すぐに保存しておいた割り込みに対して処理を始めるのです．

これと似た文に～OFF文がありますが，これは割り込みを禁止して，全く無視します．ですから，再び～ON文を実行してもその間に起った割り込みは覚えていません．

以上が割り込み処理の一般的流れですが，基本的には，呼び出し方の変わったサブルーチンだと思えば，特に難しい事ではありませんね．一番始めに挙げた例（STOPキーによる割り込み）はこの特殊な形で，BASICで，すでにSTOPキーを押した時の処理が定義されていて，STOPキーが押されたらプログラムの実行，あるいは，プログラムリストの表示を中止するという事になっています．

この様に，割り込みとは一度定義して，許可さえすれば必要な時に必要な事をしてくれる大変有難い機能ですが，使い方が多少面倒です．しかし，慣れればこれほど便利なものも無く，特にリアルタイムの処理，例えば一刻を争うゲームとか，一定時間に別のプログラムを走らせるデモプログラムとか，応用に事欠きませんし，また，割り込みを使った事で，プログラムがすっきりすることや，割り込みが不可欠な場合も多いのです．ですから割り込みを知らなかった人や，毛嫌いする人でも割り込みに慣れて，十分使いこなすことが必要だと思います．

N_{88}-BASICでは，この割り込み機能を各ハードウェアごとに幾つかサポートしていますので，1つずつ詳細に説明していきましょう．

6.2 プログラマブル・ファンクションキー

プログラマブル・ファンクションキー（以下PFキーと略します）を使った割り込み処理，という話に入る前に，このPFキーを自分なりに再定義する方法について説明しておきましょう．

6.2.1 プログラマブル・ファンクションキーの通常の機能

CONSOLE文の3番目のパラメータを覚えていますか．1を指定した時には，ファンクションキー（以下PFキーと略す）を一番下の行に表示し，0を指定した時には表示しません．このPFキー［f・4］を押すと〝LIST〟が表示され，リターンキーを押せばプログラムリストが表示されます．また［f・5］ではプログラムの実行が始まる，というなかなか便利なキーです．確かにキーを1文字づつ打つ手間が省けて便利ですが，このキーを自由に定義して，コマンドや文字列を楽に打てればなあと思いませんか？ 実はこのプログラマブル・ファンクションキー，その名の通りプログラム可能な機能キーなのです．つまり特定の手続きを「このキーで一発！」とする事が出来るのです．しかも［f・1］から［f・5］と，シフトキーを押しながらの［f・6］から［f・10］

までの10個ものキーがあるのです．これを使わない手はありません．まずはこれらのキーに何が定義されているか調べて見ましょう．

KEY LIST

と打って下さい．１文字ずつ打っているあなた！ [f・7] と [f・4] としてリターンキーの３回で良いのですよ．

```
Ok
key list
load "         save  "
auto           key
go to          print
list           edit .C/R
runC/R         contC/R
Ok
■
```

何やら表示されましたが，これがそれぞれのPFキーに定義されている文字列です．PF5の定義に見慣れない文字 C/R とありますね．実はこれはリターンキーの事です．リターンキーは入力の終りに押すキーで，打っても別に文字は表示されません．入力の終りだという特殊なキーで，一般にコントロールキャラクタ（制御文字）と呼びますが，この種のキーは，普通入力しても表示されず，また表示してもらっては困るキーなのです．しかしこの場合，それでは何が定義されているか分からないので，特別にC/Rと表示してあるのです．つまり [f・5] は [R] [U] [N] [⏎] と押したのと同じ事になるのです．

さて，何が定義されているのかは分かったので，今度は再定義をしてみましょう．

KEY 〈キー番号〉，〈文字列〉

これがキー定義をする文，KEY 文の書式です．まずは使ってみましょう．

PF1 に〝width 80″という文字を定義します．

KEY 1, 〝width 80″

これで良いのですが，「リターンキーも打つのが面倒だからこれも定義したい」と言う時には CHR$ 関数を使って下さい．

KEY 1, 〝width 80″＋chr$(13)

リターンキー等，コントロールキャラクタは，文字列に「〝」を使って直接含ませる事はできませんし，引用符「〝」も文字列を囲む時に使うので同様です．それぞれの PF キーには，最大15文字まで自由に文字列が定義できます．

KEY 文で15文字よりも長い文字列を与えた場合には，16文字目以降は無視されます．

```
Ok
key 1,"abcdefghijklmnopq"
Ok
key list

abcdefghijklmno save "
run "           files
renum           2:
list            width 80,25
run             screen ,3
Ok
■
```

また，PF キーに定義されている文字列は，CONSOLE 文の 3 番目のパラメーターを 1 にして実行すれば，表示されます．ただし，この場合15文字の内，横40字の時には 5 字，横80字の時には11字だけ表示しますので，長い文字列を定義した時には KEY LIST を実行して下さい．次のプログラム例では実行後，キー定義は BASIC を走らせた最初の状態にします．

```
100 KEY 1,"load "+CHR$(34)
110 KEY 2,"auto "
120 KEY 3,"go to "
130 KEY 4,"list "
140 KEY 5,"run"+CHR$(13)
150 KEY 6,"save  "+CHR$(34)
160 KEY 7,"key "
170 KEY 8,"print "
180 KEY 9,"edit ."+CHR$(13)
190 KEY 10,"cont"+CHR$(13)
```

6.2.2 プログラマブル・ファンクションキーを割り込みで使う

さて PF キーの定義の仕方も分かった所で，このキーのもう一つの重要な機能，キー割り込みについて説明しましょう．

まずは割り込みの宣言をしなければなりません．割り込み処理ルーチンの行番号を定義します．

ON KEY GOSUB 〈行番号〉[,〈行番号〉……]

PF キーは10個あるので，10個分の行番号を与えることができます．最初に f･1 の時の割り込み処理ルーチンの行番号，次に f･2 の行番号，と書きますが， f･1 と f･2 だけの場合は 2 つだけで結構です．また f･5 だけの場合には， f･1 から f･4 の行番号を省略します．

ON KEY GOSUB , , , , 〈行番行〉

ここで与える行番号は割り込み処理ルーチンの行番号ですが，同じ行番号を与えても別にエラーとはなりません．ただし，どのキーを押したかを知る事が出来ませんので，単に PF キーのどれかを押した割り込み処理ルーチンとなってしまいます．言い換えれば，PF キーの割り込みは， f･1 から f･10 までの10個の割り込みがあるという事で，割り込み処理ルーチンも当然最大10個まで設定可能です．

この文を実行後，割り込み処理ルーチンの行番号は，再び ON KEY GOSUB 文を実行するか，RUN が実行されるまで有効ですが，この文は行番号の定義しか行わず，この状態ではまだ割り込みは禁止状態となっています．ですから割り込みを可能とするには次に説明する KEY ON 文を実行しなければなりません．

☆KEY ON／OFF／STOP 文

割り込みの状態には次の 3 つがあります．

① **許可（ON）状態**

割り込みが発生すると，すぐさま割り込み処理ルーチンをコールし，割り込み処理を行う．

② **禁止（OFF）状態**

割り込みが発生しても無視する．処理ルーチンが定義されていても，割り込み処理は行われない．

③ **停止（STOP）状態**

割り込みが発生しても割り込み処理は行わない．しかし，無視はせず，割り込みが発生した事を覚えており，後に許可された時点で割り込み処理を始める．

実際の割り込み処理ではこの 3 つの状態が必要です．それは割り込みの発生が外部の要因に依存していて，また時間的に予測が不可能なためです．割り込みの準備が終っていないのに，割り込み処理を始めてしまっては，プログラムは意図しない結果を生じます．また割り込み処理ルーチンを，実行中に割り込みが発生するという多重割り込みも同じです．そこで BASIC での割り込み処理ではこの 3 つの状態を設定する文があります．

PF キーの割り込みを可能状態にするための

KEY （〈キー番号〉）ON

禁止するための

KEY （〈キー番号〉）OFF

そして保持させるための

KEY （〈キー番号〉） STOP

の 3 つがそうです．

この文の実行後，指定されたキーの状態が変わりますが，キー番号を省略した場合，例えば KEY ON を実行した時には，全ての PF キーの割り込みが許可されます．

前に ON KEY GOSUB 文で述べましが，この文を実行しても，割り込み処理ルーチンの行番号を宣言するだけで，割り込みは許可されません．ですから，次にKEY ON 文を実行し，割り込みを許可しなければなりません．

☆**割り込み処理ルーチン**

ON KEY GOSUB 文で指定する行番号とは，割り込み処理ルーチンの先頭行番号です．この割り込み処理ルーチンは，前にも述べた様に非常にサブルーチンと似ており，割り込みにより中断されたメインプログラムに再び戻るためには RETURN 文が必要です．

GOSUB 文でサブルーチンを呼んだ時には，行番号を覚えておくためにスタックに，この情報を積み，そして RETURN 文によってスタックから取り出されますが，RETURN 文ではなく，GOTO 文を使うとこのスタックは積まれるばかりで，いつかはメモリがなくなります．

```
new
Ok
100 *loop
110  gosub *subroutine
120 *return.line
130  goto *loop
140 '
150 *subroutine
160  goto *return.line
run
Out of memory in 110
Ok
■
```

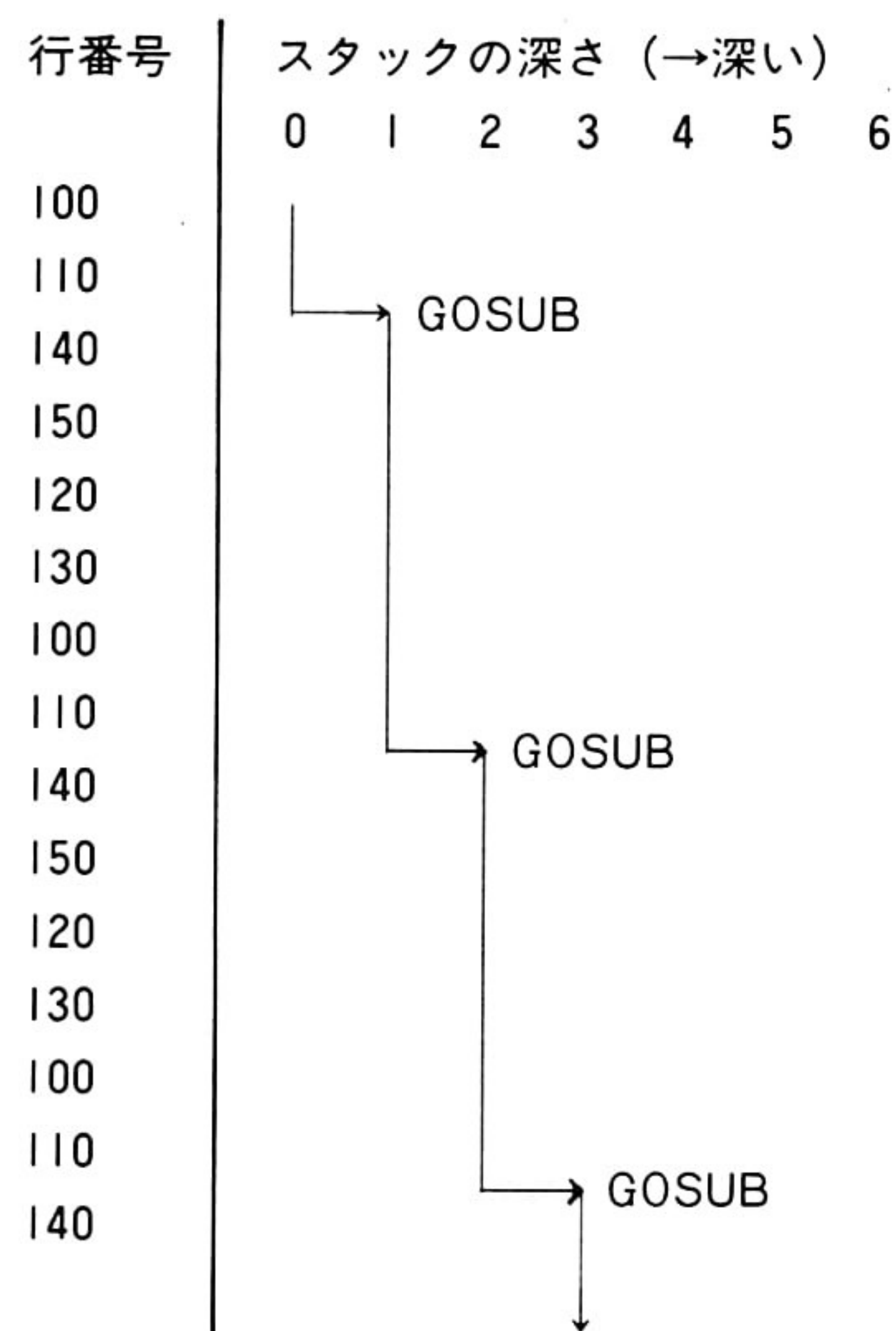
行番号	スタックの深さ（→深い）
	0 1 2 3 4 5 6
100	
110	
140	GOSUB
150	
120	
130	
100	
110	
140	GOSUB
150	
120	
130	
100	
110	
140	GOSUB

RETURN文がないので，スタックは深くなるばかりでいつかはメモリが不足する．

図3　GOSUB文とスタックの関係

これは当然の事で，ここで GOSUB 文に呼ばれている部分は，普通，サブルーチンとは呼ばれませんし，この様な使い方は滅多にしません．

割り込み処理ルーチンも呼ばれ方がサブルーチンと似ていて，割り込みのあった時点で実行していた行番号を覚えておくために，その行番号をスタックに積みます．ですから正常に実行を続けるためには RETURN 文により元の所へ戻らなければなりません．

次のプログラムは，メインルーチンでπ(円周率)の値を求めていますが，[f・1]を押すとその時まで求めた値を表示します．何度も押すと，次第に正しい値に近づくのが分かります．

```
list
1000 ' test program for key interupt
1010 '
1020 ' calculate value of pai by montecalro method
1030 '
1040 ON KEY GOSUB *DISPLAY
1050 KEY (1) ON
1060 COUNT=0 : PAICOUNT=0
1070 *LOOP
1080  X=RND : Y=RND : COUNT=COUNT+1
1090  IF (X*X+Y*Y)<=1 THEN PAICOUNT=PAICOUNT+1
1100  GOTO *LOOP
1110 '
1120 *DISPLAY
1130  PAI=PAICOUNT/COUNT*4
1140  PRINT "ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ =";PAI
1150  RETURN
OK
run
ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ = 2
ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ = 3.16854
ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ = 3.13793
ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ = 3.15058
ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ = 3.14931
```

さて，このプログラムで[f・2]を押すと再度始めから計算を始める様にしてみましょう．

```
100 ' test program for key interupt
110 '
120 ' calculate value of pai by montecalro method
130 '
140  ON KEY GOSUB *DISPLAY,*RETRY
150 *RETRY
160  KEY (1) ON : KEY (2) ON
170  PRINT "ｹｲｻﾝ ﾊｼﾞﾒ"
180  COUNT=0 : PAICOUNT=0
190 *LOOP
200  X=RND : Y=RND : COUNT=COUNT+1
210  IF (X*X+Y*Y)<=1 THEN PAICOUNT=PAICOUNT+1
220  GOTO *LOOP
230 '
240 *DISPLAY
250  PAI=PAICOUNT/COUNT*4
260  PRINT "ﾀｸﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ =";PAI
270  RETURN
```

一見正しい様ですが，何度も[f・2]を押すとOut of memory(?OM error)というエラーメッセージが表示されて止まります．これは割り込み処理ルーチンから，メインルーチンへとGOTO文により戻るためで，サブルーチンからRETURN文ではなく，GOTO文によってメインプログラムへと戻る時と同じ誤りを犯しているのです．この様な場合は割り込み処理において，しばしば起こる問題で，この解決法として次のRETURN文の特殊な使い方があります．

RETURN 〈行番号〉

この文はRETURN文とGOTO文を組み合わせた文で，普通のRETURN文と同様にスタックから戻り，行番号を取り出すのですが，そこへは戻らずに指定された行番号から実行を再開す

るのです．ですからスタックは割り込み前と同じ状態になり，プログラムは正常な動作をします．

```
100 ' test program for key interupt
110 '
120 ' calculate value of pai by montecalro method
130 '
140  ON KEY GOSUB *DISPLAY,*RETRY
150 *RESTART
160  KEY (1) ON : KEY (2) ON
170  PRINT "ｹｲｻﾝ ﾊｼﾞﾒ"
180  COUNT=0 : PAICOUNT=0
190 *LOOP
200  X=RND : Y=RND : COUNT=COUNT+1
210  IF (X*X+Y*Y)<=1 THEN PAICOUNT=PAICOUNT+1
220  GOTO *LOOP
230 '
240 *DISPLAY
250  PAI=PAICOUNT/COUNT*4
260  PRINT "ﾀﾀﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ =";PAI
270  RETURN
280 *RETRY
290  RETURN *RESTART
```

☆END 文の前に～OFF 文を

さて今度は［f･3］でその時点まで求まったπの値を表示して，実行を終らせる様にしましょう．この時にはプログラムの実行は終るので，RETURN〈行番号〉文を使わずに，いきなりEND文で良いのですが，これとは別に次の事が決まっています．

① プログラムの実行を始めた直後，割り込みは禁止状態になっている．

② 割り込みが発生し，処理ルーチンを実行し始めると，割り込みは停止（STOP）状態となり，RETURN 文により元の状態に戻る．

③ END, STOP 等，プログラムの実行が中断または終了した時でも割り込み状態はそのままである．

①はすでに説明した様に，プログラムにおいてまずON～GOSUB文により，まず割り込み処理ルーチンを定義した後，～ON文でその割り込みを許可するまで割り込みは発生しないという事です．

②はまだ説明していませんが，割り込み処理ルーチンを実行中に割り込みが発生しても，処理ルーチンへ飛ばない様，BASICが自動的に行っている事なのです．処理ルーチン内で割り込みが発生しても，RETURN文の実行後まで割り込みを保持していて，その後に割り込み処理を実行します．ですから通常の割り込み処理では，割り込みが連続して発生した場合について，頭を悩ます必要はありません．この事についてはライトペン割り込みの所で詳しく説明します．

さてここで注意したいのは③の割り込み状態がプログラム実行終了後も変化しない事なのです．

次のプログラムは［f･3］でプログラムの実行が終る様にした例ですが，実行後PFキーはどうなっているでしょう．［f･10］を押しても定義してある文字列を表示しませんが，KEY LIST を実行してみると定義はちゃんとされています．

```
110 '
120 ' calculate value of pai by montecalro method
130 '
140  ON KEY GOSUB *DISPLAY,*RETRY,*TERMINATE
150 *RESTART
160  KEY (1) ON : KEY (2) ON : KEY (3) ON
170  PRINT "ｹｲｻﾝ ﾊｼﾞﾒ"
180  COUNT=0 : PAICOUNT=0
190 *LOOP
200  X=RND : Y=RND : COUNT=COUNT+1
210  IF (X*X+Y*Y)<=1 THEN PAICOUNT=PAICOUNT+1
220  GOTO *LOOP
230 '
240 *DISPLAY
250  PAI=PAICOUNT/COUNT*4
260  PRINT "ﾀﾀﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ =";PAI
270  RETURN
280 *RETRY
290  RETURN *RESTART
300 '
310 *TERMINATE
320  GOSUB *DISPLAY
330  PRINT "ｹｲｻﾝ ｵﾜﾘ"
340  END
```

何故この様な事になるかと言うと，［f･1］から［f･3］がまだ割り込み禁止になっていない，つまりこれらのキーが割り込み要因として使われているためです．

PF キーには文字列を定義する使い方と，割り込み要因としての使い方の２つがあり，この３つの事は同時には行えません．［f･1］から［f･3］は，まだ割り込みの要因となっているので，何も表示されないのです．ですからプログラム実行後は，KEY OFF 文により割り込みを禁止しなければなりません．

```
110 '
120 ' calculate value of pai by montecalro method
130 '
140  ON KEY GOSUB *DISPLAY,*RETRY,*TERMINATE
150 *RESTART
160  KEY (1) ON : KEY (2) ON : KEY (3) ON
170  PRINT "ｹｲｻﾝ ﾊｼﾞﾒ"
180  COUNT=0 : PAICOUNT=0
190 *LOOP
200  X=RND : Y=RND : COUNT=COUNT+1
210  IF (X*X+Y*Y)<=1 THEN PAICOUNT=PAICOUNT+1
220  GOTO *LOOP
230 '
240 *DISPLAY
250  PAI=PAICOUNT/COUNT*4
260  PRINT "ﾀﾀﾞｲﾏ ﾉ pai ﾉ ｱﾀｲ =";PAI
270  RETURN
280 *RETRY
290  RETURN *RESTART
300 '
310 *TERMINATE
320  GOSUB *DISPLAY
330  PRINT "ｹｲｻﾝ ｵﾜﾘ"
340  KEY OFF
350  END
```

さて PF キーを中心に，割り込み処理についての説明をしましたので，次にこの便利な割り込みを実際に使ってみましょう．

6.2.3 プログラマブル・ファンクションキー割り込みの用途

次のプログラムは PF キーを使った簡単なゲーム，スロットマシンゲームです．［f･5］を押

すと３桁の数字が変化しますので，［f･1］から［f･3］のキーを押して，それぞれの桁の数字を止めていきます．すべて止まったら，組み合わせにより点数が加算されます．

```
1000 DEFINT A-Z : DIM F(2),PF(2),L(2)
1010 COLOR 7,0,0,7 : CONSOLE 0,20,0,1 : WIDTH 40,20
1020 LOCATE 14,10 : PRINT "ﾁｮｯﾄ ﾏｯﾃﾈ｡"
1030 SCREEN 0,3 : CLS 2
1040 READ P$ : N=LEN(P$) : P$=P$+P$
1050 L(0)=1 : L(2)=1 : L(1)=2 : MONEY=50
1060 ON KEY GOSUB *KEY1,*KEY2,*KEY3,*BREAK,*START
1070 '
1080 GOSUB *DISPLAY
1090 KEY (5) ON
1100 *LOOP
1110  IF PF=0 THEN *LOOP
1120  FOR I=0 TO 2:KEY STOP
1130   IF F(I) THEN:PF(I)=PF(I)MOD N+1:FOR J=0 TO 2:COLOR L(J)
                 :LOCATE 15+I*3,J+6:PRINT MID$(P$,PF(I)+J,1);:NEXT
1140  KEY ON:NEXT
1150  GOTO *LOOP
1160 '
1170 '
1180 *KEY1 : KK=0 : GOTO *KEYINT
1190 *KEY2 : KK=1 : GOTO *KEYINT
1200 *KEY3 : KK=2
1210 *KEYINT
1220  IF F(KK)=0 THEN RETURN
1230  F(KK)=0 : PF=PF-1
1240  IF PF THEN RETURN
1250  COLOR 7 : COUNT=0
1260  IF PF(0)=PF(1) THEN COUNT=COUNT+1
1270  IF PF(0)=PF(2) THEN COUNT=COUNT+1
1280  IF PF(1)=PF(2) THEN COUNT=COUNT+1
1290  LOCATE 3,15
1300  ON COUNT+1 GOSUB *LEVEL0,*LEVEL1,*LEVEL1,*LEVEL2
1310  MONEY=MONEY+GETM
1320  LOCATE 14,8 : PRINT "ﾓﾁﾃﾝ = "MONEY
1330  IF MONEY<>0 THEN KEY OFF : KEY (5) ON : RETURN *LOOP
1340  LOCATE 5,16:COLOR 2:PRINT" ｻﾞﾝﾈﾝﾃﾞｽｶﾞ ﾓｳ ﾓﾁﾃﾝ ｶﾞ ｱﾘﾏｾﾝ"
1350  COLOR 7:KEY OFF
1360 *DUMY : IF INKEY$<>"" THEN *DUMY ELSE END
1370 '
1380 *LEVEL0
1390  PRINT "ﾊｽﾞﾚ! ﾊｽﾞﾚ!"
1400  GETM=0
1410  RETURN
1420 *LEVEL1
1430  GETM=1+RND*3
1440  PRINT "ｻﾞﾝﾈﾝ ﾃﾞｼﾀ ｶﾜｲｿｳﾀﾞｶﾗ "GETM" ﾃﾝ ｱｹﾞﾏｼｮｳ"
1450  RETURN
1460 *LEVEL2
1470  GETM=300+RND*300
1480  PRINT "ﾏｲｯﾀ! ﾏｲｯﾀ!"GETM" ﾃﾝ ﾔﾙｮ! ﾓｯﾃｸ ﾄﾞﾛﾎﾞｳ"
1490  RETURN
1500 '
1510 '
1520 *START
1530  SCREEN ,0 : CLS : LOCATE 13,4 : PRINT "slot machine"
1540  MONEY=MONEY-2 : IF MONEY<0 THEN MONEY=0
1550  FOR I=0 TO 2 : F(I)=-1 : PF(I)=RND*N+1 : NEXT
1560  PF=3 : KEY OFF
1570  KEY(1) ON : KEY(2) ON : KEY(3) ON : KEY(4) ON
1580  RETURN *LOOP
1590  '
1600  DATA "♠¥♥♣♦$○●"
1610 '
1620 *DISPLAY
1640  TILE$=CHR$(&HAA)+CHR$(0)+CHR$(&H22)+CHR$(&H55)+CHR$(0)+CHR$(&H44)
1650  LINE (235,58)-(358, 90),3,B : PAINT (300,80),TILE$,3
1660  LINE (240,69)-(255,78),0,BF
1670  LINE (288,69)-(304,78),0,BF
1680  LINE (337,69)-(353,78),0,BF
```

```
1690  LINE (200,50)-(400,100),3,B : PAINT(201,51),3,3
1700  LINE (180,101)-(420,101),4 : LINE -(500,140),4
1710  LINE -(100,140),4 : LINE -(180,101),4
1720  PAINT (300,110),4,4 : CLS
1730  LOCATE 13,4 : PRINT "slot machine"
1740  LOCATE 5,14 : PRINT " key-5 ヲ オスト slot machine ガ ハジマリマス。"
1750  PRINT " key-1 カラ key-3 デ スロット ガ ソレゾレ トマリマス。"
1760  PRINT " モシ program ヲ トメルトキ ハ key-4 ヲ オシテクダサイ。"
1770  LOCATE 8,18 : PRINT "マヅハ key-5 ヲ オシテ クダサイ。"
1780  RETURN
1790 '
1800 *BREAK : COLOR 7 : KEY OFF : END
```

6.3 HELPキー

6.3.1 HELPキーの通常の機能

機能キーは PF キーだけではありません．HELP キーがあります．このキーはプログラムの実行でエラーが起こった時に押すと，エラーの発生した行を表示し,カーソルが，誤った使い方をした文，または関数の付近に表れます．またプログラム入力中に押すと，一つ前に入力した行が表示されます．つまり，HELP キーとはエラーが起きて困った時に押すキーの意味なのですが，困った時と言ってもエラーとは限りません．「前に作ったプログラムが見つかったけど，使い方を忘れてしまった」なんて事はありませんか？「他の人に使ってもらえるプログラムを作ったけど，使い方を教えるのが大変だ！」とか．こんな時に，「困ったら HELP キーを押せ！」とだけ説明しておき，押すと使い方の説明が表示されるというのはどうでしょう．

MON コマンドで，機械語モニタへと移ってみて下さい．

MON ↲

さて，どう使うのだろうと考える前に，まず HELP キーを押すのです．すると何と，コマンドとそのパラメータの指定の仕方が表示されますね．

```
Monitor has following commands.
a <address>                     assemble source text lines.
bh or bq                        select radix (hexa decimal or octal).
d <start>,<end>                 dump contents of memory.
e <start>                       change contents of memory by screeen editor.
f <start>,<end>,<const>         fill memory by the constant.
g <start>,<break1>,<bread2>     execute user program.
i <port>                        read input port.
l <start>,<end>                 dis-assemble program in memory.
m <start:s>,<end:s>,<start:d>   move contents of memory block(s=src,d=dest).
o <port>,<new value>            output new value to the port.
p                               toggle print switch.
r <file name>                   read cassette file.
s <address>                     substitute memory contents.
tm                              test memory.
v <file name>                   verify cassette file.
w <file name>,<start>,<end>     write cassette file.
x or x <register name>          dump all CPU registers or change the register.
CNTL-b                          return to BASIC.
CNTL-d <dr>,<sur>,<tr>,<sec>    dump contents of sectors.
              [,<tr>,<sec>]     <sur> is optional.
CNTL-r <dr>,<sur>,<tr>,<sec>    read sectors into memory.
             ,<start>,<end>
CNTL-w <dr>,<sur>,<tr>,<sec>    write contents of memory into sectors.
             ,<start>,<end>
h]■
```

ここではモニタを扱わないので，BASICへ戻るコマンド，CTRL-Bで戻りますが，こんなHELPキーの使い方，なかなか賢いと思いませんか？ 私などこのモニタでのHELPキーの使い方，思わず「すごい！」と言ってしまったほどです．これを応用しない手はありません．

6.3.2 HELPキーを割り込みで使う

さてこの様にHELPキーには困った時に押すキー，お助けキーとしての役割が考えられ，プログラムの説明が表示されるとか，入力の仕方が分かるとか，いろいろ用途がありますが，ただ押せば良いのではありません．HELPキーにはエラー箇所を表示してくれると言う，便利な機能がありますが，これはBASICが持っている機能なのです．ですからあなたがこれとは別の事をさせるためには，あなた自身がプログラムで，その機能を実現しなければなりません．そこで，このためにHELPキーには割り込み機能があるのです．HELPキーの割り込み処理ルーチンで，別の機能を実現するのです．

HELPキーの割り込みは，PFキーの場合とほとんど同じです．ですからPFキー割り込みを理解していれば，すぐにもHELPキー割り込みを使いこなせます．

☆ON HELP GOSUB 文

HELPキー割り込みでも，まず割り込み処理ルーチンを宣言しなければなりません．PFキーの場合には，キーが10個あるので，行番号も必要なだけ指定しましたが，HELPキーでは指定する行番号は1つだけです．

ON HELP GOSUB 〈行番号〉

また，ON KEY GOSUB文と同様に，この文を実行しても，HELPキー割り込みの状態は変わりません．ですから，HELP割り込みを許可するためには，次のHELP ON文を実行しなければなりません．

またON HELP GOSUB文で指定する行番号はプログラム中に存在する行番号でなければなりませんが，いったん宣言すれば，再宣言するか，RUNを実行するまで有効です．

☆HELP ON／STOP／OFF

HELPキー割り込みにも割り込みの状態が3つあり，この文を使ってHELP割り込みを許可(ON)，停止(STOP)，禁止(OFF)のいずれかに指定します．プログラム実行開始時には，HELP OFFの状態になっているので割り込み機能を使う時は，ON HELP GOSUB文の実行後，HELP ONを実行しなければなりません．

6.3.3 HELPキー割り込みの用途

HELPキー割り込みは，PFキー割り込みと使い方も動作もほとんど同じです．それなら何も

HELP キーを使わなくても，PF キーの割り込みを使っても良いではないかと思えてきませんか？確かに HELP キーでも PF キーでも一見同様に思えますが，1つ違う事があります．INPUT 文などのキーボードからの入力命令実行中には，PF キー割り込みは禁止されているのです．

☆INPUT 文実行中に HELP キーで割り込む

プログラムでこの HELP キーを使うのは，「INPUT 文実行で入力待ちになったけれど，何を入力すれば良いか分からない」という時なのです．この時，PF キー割り込みは，禁止されているので役に立ちません．そこで禁止されない HELP キー割り込みを使うのです．この用途には HELP キーが大変役に立ちます．

☆再び INPUT 文に戻るには

さて，INPUT 文実行中に HELP 割り込みが発生したので，割り込み処理ルーチンで，入力方法の説明を表示し，RETURN 文で割り込み処理を終えたとします．その時，実行を再開するのは，INPUT 文ではなく，その次の文なのです．ですから再度入力待ちにするためには，RETURN 〈行番号〉 文を使うか，または何か適切な処置をしなければなりません．

☆強制的に入力文に戻す方法

```
100 ON HELP GOSUB *MESSAGE : HELP ON
110 '
120 *LOOP
130  INPUT "HH:MM:SS ";T$
140  IF LEN(T$)=0 THEN *BREAK
150  HOUR=VAL(LEFT$(T$,2))
160  MINUTE=VAL(MID$(T$,4,2))
170  SECOND=VAL(MID$(T$,7,2))
180  TOTAL=(HOUR*60+MINUTE)*60+SECOND
190  PRINT TOTAL" 秒"
200  GOTO *LOOP
210 '
220 *BREAK : HELP OFF : END
230 '
240 *MESSAGE
250  PRINT
260  PRINT "00時00分00秒 ｶﾗ ｹﾞﾝｻﾞｲ ﾏﾃﾞﾉ 秒ｽｳ ｦ ﾓﾄﾒﾏｽ"
270  PRINT "ｹﾞﾝｻﾞｲ ﾉ 時 分 秒 ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
280  PRINT "ﾓｼ ﾄﾒﾙﾄｷﾊ return ｷｰ ﾀﾞｹｦ ｵｼﾃｸﾀﾞｻｲ"
290  RETURN *LOOP
```

このプログラムでは，HELP 割り込みの処理ルーチンからの復帰に，RETURN 〈行番号〉 文を使っています．これは，HELP 割り込みが発生するのは入力待ちの時だけを仮定していて，さらに INPUT 文が1つしかないプログラムだから可能な事です．しかし，INPUT 文は一つとは限りませんし，割り込みもいつ発生するか分かりません．

☆フラグを使う方法

そこで次のプログラムの様に，ある変数を，HELP 割り込み処理を実行したと言うフラグにし，

INPUT 文実行後はそのフラグを調べる方法があります.

```
100 '
110 ' help ﾜﾘｺﾐ sample program
120 '
130 FLAG=0 : ' if help then flag=-1 else flag=0
140 ON HELP GOSUB *MESSAGE : HELP ON
150 '
160 *LOOP
170  FLAG=0 : INPUT "x = ";X
180  IF FLAG THEN 170
190  FLAG=0 : INPUT "y = ";Y
200  IF FLAG THEN 190
210  FLAG=0
220  IF X=0 AND Y=0 THEN *BREAK
230  PRINT "(0,0)-("X","Y") = "SQR(X*X+Y*Y)
240  GOTO *LOOP
250 '
260 *BREAK : STOP OFF : END
270 '
280 *MESSAGE
290  PRINT
300  PRINT "    ﾍｲﾒﾝ ｼﾞｮｳﾃﾞ ｹﾞﾝﾃﾝ (0,0) ｶﾗ ｱﾙｲﾁ (x,y) ﾏﾃﾞﾉ ｷｮﾘｦ ｹｲｻﾝｼﾏｽ"
310  PRINT "            x ﾄ y ﾆ ｱﾀｲｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
320  PRINT "            ﾓｼ ﾔﾒﾙﾄｷﾊ ﾄﾞﾁﾙﾗﾓ 0 ﾆｼﾃｸﾀﾞｻｲ"
330  FLAG=-1 : RETURN
```

6.4 STOPキー

6.4.1 STOPキーの通常の機能

プログラムの実行や，リストの表示を止めるキーとしておなじみの STOP キーにも，割り込み機能があります．この STOP キーによる割り込みも，PF キーや HELP キーの割り込みと使い方は同じですので，詳しくは PF キーの割り込み処理の所を参照して下さい．ここでは簡単に割り込み宣言の仕方を説明しましょう．

6.4.2 STOPキーを割り込みで使う

☆ON STOP GOUSB 文

STOP キー割り込みを使うには，まずこの文を実行しなければなりません．

ON STOP GOSUB <行番号>

この文により，STOP 割り込み処理ルーチンを行番号を指定して定義しますが，当然指定する行番号は，プログラム中に存在しなければなりません．またこの文では，割り込みの状態を変えないので，次の STOP ON 文を実行しなければならないのは，PF キーや HELP キーと同じです．

☆STOP ON／OFF 文

この文により，STOP 割り込みの状態を変えます．割り込み機能を使う時には，ON STOP GOSUB 文の次に，STOP ON 文を実行するのは言うまでもありません．

割り込みについては，PF キー割り込み，HELP キー割り込みについて，説明していますので，分からない時には，PF キー割り込みについてもう一度読み返して下さい．

6.4.3 STOPキー割り込みの用途

☆STOP キーの機能を失わせる（中断不可能にする）

PF キー，HELP キーとキー割り込みについて説明してきましたが，これらのキーには，どれも二通りの使い方があったのを覚えていますか？

一つは割り込みを禁止した使い方で，この時には，これらのキーの通常の機能が働きました．文字例を定義するとか，エラー発生箇所を表示するのがそうです．

もう一つは割り込み要因としての使い方で，この時には，これらのキーが押されると定義されている割り込み処理ルーチンを実行しましたね．

そして，この二つの使い方は同時には行えませんでした．割り込みの要因として使った場合には，通常の機能は失われましたね．

STOP キーにおいてもこの二つの使い方があり，通常の使い方は，プログラムの実行やリストの表示を中断する事です．そしてもう一つの使い方，STOP キーを割り込み要因として使う場合には，やはり通常の機能は失われます．

実は STOP キー割り込みの最大の利点は，この通常の機能が失われる事にあるのです．

```
100 ON STOP GOSUB *BREAK
110 PRINT "STOP=OFF"
120 FOR DELAY=0 TO 1000 : NEXT
130 PRINT "STOP=ON" : STOP ON
140 FOR DELAY=0 TO 1000 : NEXT
150 PRINT "STOP=OFF" : STOP OFF
160 END
170 '
180 *BREAK
190  PRINT "STOP ｷｰ ﾊ ﾜﾘｺﾐ ｷｰ ﾆ ﾅｯﾃｲﾏｽ"
200  RETURN
```

このプログラムでは，STOP 割り込みの状態を OFF，ON，OFF と変えていますが，STOP キーでプログラムを中断できるのは STOP OFF の状態だけです．つまり，STOP 割り込みを使う事で，プログラムを止められなくする事が出来るのです．実際のプログラムでも，このルーチンを実行している時は，STOP キーで中断されては困る，という場合があります．こんな時に STOP キーを割り込みに使うのです．また，この極端な使い方として，プログラムを実行終了まで止められなくする事も出来ます．

```
100 CONSOLE ,,0,1 : WIDTH 40,20
110 ON STOP GOSUB *BREAK
120 '
130 PRINT "ﾀﾀﾞｲﾏ "TIME$" ﾃﾞｽ "
140 PRINT "ｵﾒｻﾞﾒﾉ ｼﾞｺｸｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
150 INPUT "HH:MM:SS";T$
160 STOP ON
170 CLS
180 LOCATE 10,10 : PRINT "ﾀﾀﾞｲﾏ ﾉ ｼﾞｺｸ ";
190 '
200 *LOOP
210  LOCATE 23,10 : PRINT TIME$
220  IF T$<>TIME$ THEN *LOOP
230  STOP OFF : CLS
240  FOR I=0 TO 100
250   FOR COL=1 TO 7
260    COLOR COL
270    BEEP 1 : PRINT "ｼﾞｶﾝ ﾀﾞﾖ !!" : BEEP 0
280   NEXT
290  NEXT
300  COLOR 2
310  PRINT "ﾓｳ ｼﾗﾅｲﾖ !!!"
320  COLOR 7
330  END
340 '
350 *BREAK : RETURN
```

このプログラムでは PC-8801 の持っている CLOCK 機能を使ってデジタル目ざまし時計を実現していますが，一度時刻をセットすると指定した時刻になるまでは，STOP キーまたは CTRL-C のいずれによってもプログラムは止まりません．この例の他にも，デモプログラムの実行中に，STOP キーで勝手に止められない様にするという用途もあります．

☆中断した時にユーザー独自の機能を付加する

また，こんな使い方もあります．プログラムの実行を STOP キーで止めた時に音が出ますね．この音，さほど大きな音ではありませんが，夜中に使っていたりすると気になりませんか？ こんな時には，次の様にしておくのです．

```
100 ON STOP GOSUB *BREAK : STOP ON
110 '
120 FOR I=0 TO 359
130  R=I*3.14159/180
140  PRINT USING " sin(###)=+#.##### : cos(###)=+#.##### ";
                  I,SIN(R),I,COS(R)
150 NEXT
160 '
170 *BREAK : STOP OFF : END
```

これで，STOP キーを押すとプログラムの実行は終り，しかもうるさい音も出ませんね．ただこの場合には，プログラムの中断が，どこで起こったか分かりませんが．

その他，プログラム中で，カーソルを消していたり，スクロール・ウィンドウを狭くしていたり，PF キー割り込みを使っていたり，パレットを変更していたりしている場合，単に STOP キーで実行を止めてしまうと，それらは初期状態に戻らぬまま，コマンドレベルに戻ってしまいます．例えば，中断してプログラムを変更しようとしても非常に困難です．

このような時，STOP キーの割り込みルーチンに，上記したような設定を，全て初期化する命

令を書いておけば，すぐに編集作業に移れて，とても便利です．

6.4.4 STOPキー割り込みの注意

この様に，STOP キー割り込みには，割り込み処理を起動する事よりも，プログラムの中断を禁止する事に大きな利点があります．しかし，この使い方を間違えると，とんでもない事になります．

プログラムのデバックが，まだ終っていないのにSTOP キー割り込みを使っている場合がそうです．エラーで止まるバグなら良いのですが，無限に繰り返すループにはまり込んでいたりすると，これはもう STOP キーをいくら押しても止まりません．ですから STOP キー割り込みを使う時は，くれぐれも注意して下さい．

万一，この様な場合が起った時には，こんな方法しかありません．

リセットボタンを押すか，電源をいったん切る事で BASIC の初期状態からやり直す．

この方法ではプログラムが消えてしまうので，次の方法をおすすめします．

STOP キーを押しながらリセットボタンを押す．

これにより BASIC はプログラムが残った状態で再起動します．

6.5 タイマ

6.5.1 タイマの通常の機能

PC-8801 にはカレンダークロックが内臓されており，日付と時刻を設定して，時計として利用する事が出来ます．この設定は，時刻を TIME$, 日付を DATE$ を用いてそれぞれ，“HH：MM：SS”の型，“YY／MM／DD” の型の文字列を与えます．

例）現在の日付と時刻を1982年 7 月 4 日正午に設定する．

```
DATE$= "82/07/04"
TIME$= "12:00:00"
```

一度設定すれば，以後この TIME$ と DATE$ を使って現在の日付と時刻を参照できますが，PC-8801 では，このクロックをバッテリーによって，バックアップしていますので，電源を切った場合でも，日付，時刻は失われません．

次のプログラムでは，これを使って時計プログラムを実現しており，毎時 0 分に時報を出します．

```
100 CONSOLE 0,20,0,1 : WIDTH 40,20
110 *LOOP : IF TIME$=T$ THEN *LOOP
120   T$=TIME$ : M=VAL(MID$(T$,4,2)) : S=VAL(MID$(T$,7,2))
130   IF M=0 AND S=0 THEN BEEP 1
140   LOCATE 14,10 : PRINT T$ : BEEP 0
150   GOTO *LOOP
```

さて，時計機能を使って目ざまし時計を作ってみましょう．これは IF 文で，常に時刻を調べる事で簡単に実現出来ます．

```
100   CONSOLE 0,20,0,1 : WIDTH 40,20
110   PRINT "ｲﾏ "TIME$
120   INPUT "ｵﾒｻﾞﾒ ﾉ ｼﾞｺｸ (HH:MM:SS) ";W$
130   CLS
140  *LOOP : IF TIME$=T$ THEN *LOOP
150   T$=TIME$
160   LOCATE 14,10 : PRINT T$
170   IF T$<>W$ THEN *LOOP
180   LOCATE 6,12 : PRINT "ｼﾞｶﾝ ﾀﾞｮ  ﾅﾆｶ ｷｰｦ ｵｼﾃﾈ" : BEEP 1
190   *KEYWAIT : LOCATE 14,10 : PRINT TIME$
200    IF INKEY$="" THEN *KEYWAIT
210   BEEP 0
220   END
```

さて，このプログラムをさらに拡張して，３分間経つと時間切れになるゲームを作ろうと思っても，簡単にはいきません．プログラムのいたる所で現在の時刻を調べなければならないのです．へたをするとゲームの実行速度を大幅に低下させることにも成り兼ねません．

6.5.2　タイマを割り込みで使う

N_{88}-BASIC ではこの時計機能により，タイマ割り込みが実現されています．ですからこんな場合には，この割り込み機能を使って，指定した時刻で割り込みを発生する事が出来ます．

☆ON TIME$ GOSUB 文

この文はタイマ割り込みの発生時刻と，その時に分岐する割り込み処理ルーチンの行番号を定義します．

ON TIME$= "HH : MM : SS" GOSUB <行番号>

この文の実行で，割り込み時刻が設定されますが，この時には，指定された時刻から現在の時刻を引き算して割り込み時刻を設定します．したがって，現在の時刻を設定しなおしてから，割り込み時刻を宣言する場合，TIME$ 文は必ず ON TIME$ GOSUB 文の前に実行しなければなりません．

また指定された時刻から現在の時刻を引き算した結果は，内部では 16bit 整数に圧縮されます．１日は86400秒ですが，16bit で表す事が出来るのは65535までですから，完全に１秒単位で処理することができません．したがって，この ON TIME$ GOSUB 文で制御できる時刻の分解能は２秒で，場合によっては，設定した時刻と実際に割り込みが発生する時刻との間に，±１秒程度の誤

差が生じることがあります．このことは，秒単位でのシビアーな割り込みはできないことを意味します．

☆TIME$ ON／STOP／OFF

この文によりタイマ割り込みの状態を設定します．

割り込み発生のためには，TIME$ ON を実行して下さい．

6.5.3 タイマ割り込みの用途

この割り込みの応用としてまず思い付くのは，先の目ざまし時計ですが，こんな使い方はどうでしょう．

① 決められた時間内で行うゲーム

入力待ちになったからと言って，落ちついて考え込んでいる時でも，時計は待ってくれません．まさに，一刻を争うゲームを作ってみませんか？

② プログラマブル・クロック

ビデオデッキやオーディオタイマなどで，１週間番組予約機能というのがありますね．これを実現するのです．あらかじめ時刻を指定しておくと，「ニュースの時間ですよ！」とか，またテレビやラジオの番組に限らず，一日の予定を覚えさせておき，「お客さまのみえる時間ですよ！」と知らせてくれるのはどうでしょう．

まだまだほかにも考えられますが，ここでは簡単に，割り込み機能を使って，ローマ数字デジタル時計を作って見ました．実行させて使い方が分からない時は，HELP キーを押して下さい．

```
100 '
110 ' time ﾜﾘｺﾐ / sample program
120 '
130 DEFINT A-Z : DIM P$(12),D$(7) : BUZ=0
140 ON HELP GOSUB *MESSAGE
150 CONSOLE 0,20,0,1 : WIDTH 40,20
160 FOR I=0 TO 12 : READ P$(I) : NEXT
170 D$(1)=" 時" : D$(4)=" 分 " : D$(7)=" 秒"
180 HELP ON
190 '
200 *LOOP
210  IF T$=TIME$ THEN *LOOP
220  IF BUZ THEN BP=NOT BP : IF BP THEN BEEP 0 ELSE BEEP 1
230  T$=TIME$ : H=VAL(LEFT$(T$,2)) MOD 12 : IF H=0 THEN H=12
240  D$(0)=P$(H)
250  D$(2)=P$(VAL(MID$(T$,4,1)))
260  D$(3)=P$(VAL(MID$(T$,5,1)))
270  D$(5)=P$(VAL(MID$(T$,7,1)))
280  D$(6)=P$(VAL(MID$(T$,8,1)))
290  LOCATE 3,10
300  FOR I=0 TO 7 : PRINT D$(I); : NEXT
310  IF BUZ THEN IF INKEY$<>"" THEN BEEP 0 : BUZ=0 : CLS
320  GOTO *LOOP
330 '
340 DATA "     ","    I","   II","  III","   IV","    V","   VI"
350 DATA "  VII"," VIII","   IX","    X","   XI","  XII"
360 '
370 *MESSAGE
```

```
380  CLS
390  PRINT "ｺﾚﾊ ﾛｰﾏ ｽｳｼﾞﾃﾞ ﾋｮｳｼﾞｼﾀ degital clock ﾃﾞｽ｡"
400  PRINT "ｺﾉ ﾄｹｲ ﾆﾊ alarm ｷﾉｳｶﾞ ｱﾘﾏｽ｡"
410  PRINT "ｼﾃｲｼﾀ ｼﾞｺｸﾃﾞ ﾌﾞｻﾞｰｦ ﾅﾗｼﾏｽ｡"
420  PRINT "ﾅﾝｼﾞﾆ ｼﾏｽｶ ? ( ﾓｼ ｼﾃｲｼﾅｲ ﾄｷﾊ return ｷｰ ﾀﾞｹｦ ｳｯﾃｸﾃﾞｻｲ｡"
430  INPUT "(HH:MM:SS)";WAKEUP$
440  IF WAKEUP$<>"" THEN ON TIME$=WAKEUP$ GOSUB *WAKEUP : TIME$ ON
450  CLS
460 RETURN
470 '
480 *WAKEUP
490  BUZ=-1 : BP=-1
500  LOCATE 5,7 : PRINT "ﾅﾆｶ ｷｰｦ ﾀﾀｸﾄ ﾌﾞｻﾞｰ ﾊ ﾄﾏﾘﾏｽ｡"
510 RETURN
```

6.6 ライトペン

6.6.1 ライトペンとは

テレビ番組などで，コンピュータのビデオディスプレイに文字や図形を表示させておき，万年筆の様なもので画面をつついて，指示を与えているのを見た事がありませんか？　コンピュータを使ったことのない人でも，このようにライトペンを使っているのを見ると，それだけでコンピュータとはこんなものかと感心してしまうかもしれません．

このライトペンは，何も高級な装置という訳ではなく，パーソナルコンピュータでも十分利用できるのです．

PC-8801 の場合でも，このライトペンを手軽に使う事ができますが，残念ながら標準装備ではなく，必要な場合に買い求めるオプションとなっています．しかし BASIC では，すでにライトペンに関するステートメントや関数が用意されていますし，また接続方法も簡単です．

6.6.2 ライトペンから情報を得る

ライトペンは計算機に指示を与える方法として，キーボードから入力する方法と違って，なかなか便利です．N_{88}-BASIC には，このライトペンのための関数として PEN 関数があります．

PEN (〈機能〉)

この関数は，〈機能〉の値により返す結果が異なります．

☆PEN (0)

この関数は，ライトペンを押したら－1を，押されていないと0を返します．また一度押されると－1になりますが，後述する ON PEN GOSUB 文で定義した割り込み処理ルーチンに入るまでこの値を保持します．そして処理ルーチンに入った時点で0となります．

☆PEN (1)

ライトペンが押された時の，ペンの指示した画面上の位置の水平座標が，キャラクタ座標で与えられます．ですから，返す値は0から最大79の範囲ですが，画面のはじめの方では，この範囲を越える事が時としてありますので注意して下さい．

☆PEN (2)

PEN (1) と同じですが，今度は垂直位置がキャラクタ座標で与えられます．

このPEN関数を使ってライトペンの押された位置を知るのですが，注意してほしいのは，PEN関数が値を正しく返すのは，PEN ON の状態である点です．ですから PEN 関数を使う前には後述する PEN ON 文が実行されていなければなりません．

6.6.3 ライトペンを割り込みで使う

PEN 関数は PEN ON を実行して使うと説明しましたが，普通ライトペンを使う時には，割り込みを用います．ではまず，ライトペン割り込みの宣言の仕方を説明しましょう．

ライトペン割り込みを使う時にも，次の文を実行して処理ルーチンの行番号を定義しなければなりません．

ON PEN GOSUB 〈行番号〉

これにより処理ルーチンは宣言されますので，後はライトペン割り込みを許可するだけです．

PEN ON／OFF／STOP

この文によりライトペン割り込みを，許可，停止，禁止します．割り込みについては，PF キー割り込みから読んできた方には，もう十分お分かりと思いますので，ここでは詳しくは触れません．ライトペン割り込みもキー割り込みのキーを PEN に変えただけですから，すぐに分かると思います．

☆サンプルプログラム

さて，とにかくまずライトペンを使ってみましょう．次のプログラムを実行すると，ライトペンを押すとそのキャラクタ座標を表示し，その位置に「○」を表示します．

```
100 CONSOLE ,,0,1 : SCREEN 0,0 : COLOR 1,7,0,7 : CLS 3
110 ON PEN GOSUB *ENTPEN
120 PEN ON
130 *LOOP : GOTO *LOOP
140 '
150 *ENTPEN
160  X=PEN(1) : Y=PEN(2)
170  LOCATE 0,0
180  PRINT "pen : x="X" ; y="Y
190  LOCATE X,Y : PRINT "o";
200  RETURN
```

このプログラムを実行すると，まず画面が白くなります．これはライトペンが画面からの光を検出して働くためで，画面が黒いとライトペンは押された位置を知る事ができません．そこでライトペンで指定する画面上の領域は，すでに何か表示されていなければなりません．その時に画面の色は白でなくとも，緑でも青でもかまいませんが，一般にライトペンは赤に対する感度が，非常に悪いので注意して下さい．

6.6.4　ライトペン割り込みの注意

ライトペン割り込みを使っていると，時にはこんな事が起こります．

ライトペンを一度だけ押したつもりなのに二度割り込みが発生してしまって，うまくプログラムが動いてくれない．

この原因は，ライトペンを押すとライトペン内部にあるスイッチが押され，これで割り込みを発生するのですが，このスイッチが二度押されたかの様に，わずかの時間に連続して ON，OFF する事があるためです．これはチャタリングと呼ばれるハードウェアから起こる困った症状ですが，これをプログラムでなくす事ができます．割り込み処理ルーチンについて思い出して下さい．．割り込みが許可されている時には，割り込みが発生するとその処理ルーチンを実行し始めますが，この処理を始める時には割り込みは自動的に停止状態になりましたね．この状態の時には，割り込みは発生しても保存されました．

このチャタリングによる連続した割り込みの場合，１度目の割り込みで処理ルーチンへと分岐し，２度目の割り込みは保持されていて，このために困った問題が生じるのです．ですから，保持している割り込みを忘れさせれば良い訳です．そこでライトペン割り込みの処理ルーチンの最後で一度 PEN OFF を実行させてください．この文によって，保持していた割り込みが消去されてチャタリングを打ち消すことができます．

ただし，処理ルーチンの最後の RETURN 文を実行する直前に再び PEN ON の状態にしないと２度と割り込みがきかなくなりますよ．

6.6.5　ライトペンを使った応用例

BASIC を使いこなすコツは，とにもかくにも使うことです．ある程度理解したらプログラムを作って実行させてみましょう．そうすれば，おのずと道は開けます．

「論より RUN」を肝に命じて，まずはライトペンを使って項目を選択してみましょう．

☆項目を選択する

まずは，次のプログラムを入力し，実行してみて下さい．

```
100 '
110   DEF FNX=RND*640 : DEF FNY=RND*200 : DEF FNCOL=RND*6+1
120   SCREEN 0,0:COLOR 0,0,0,7
130   CONSOLE 0,25,0,0:WIDTH 40,25:CLS 2
```

```
140  ON PEN GOSUB *PENINT
150  PEN ON
160 '
170  LOCATE 16,20:PRINT "< MENU >" : COLOR 4
180  FOR I=0 TO 3
190   READ MENU$(I)
200   LOCATE I*9+2,22:PRINT SPC(6);
210   LOCATE I*9+2,23:PRINT MENU$(I);
220   LOCATE I*9+2,24:PRINT SPC(6);
230  NEXT I
240  COLOR 0
250  *LOOP : GOTO *LOOP
260 '
270 *PENINT
280  PX=PEN(1):PY=PEN(2)
290  IF PY<22 THEN *EXIT
300  BEEP 1 : FOR DELAY=1 TO 10:NEXT : BEEP 0
310  IF  1<PX AND PX< 8 THEN GOSUB *BOX : GOTO *EXIT
320  IF 10<PX AND PX<17 THEN GOSUB *CIRCL : GOTO *EXIT
330  IF 19<PX AND PX<26 THEN GOSUB *CRTCLS : GOTO *EXIT
340  IF 28<PX AND PX<35 THEN GOTO *BYE
350 *EXIT:PEN OFF:PEN ON:RETURN
360 '
370 *BOX
380  LINE(FNX,FNY)-(FNX,FNY),FNCOL,B
390  RETURN
400 '
410 *CIRCL
420  CIRCLE (FNX,FNY),RND*100+40 FNCOL
430  RETURN
440 '
450 *CRTCLS
460  CLS 2:RETURN
470 '
480 *BYE
490  PEN OFF:END
500 '
510 DATA " box  ",circle,"clear ","  end  "
```

ライトペンを使ってメニューを選択するには，まず画面上にその項目を表示します．次に表示した位置を考えて，領域を分けます．後は PEN 関数でライトペンが押された位置を読み取り，対応する処理をするだけです．

ここでは，ライトペンで画面上に表示する図形を指定しています．

☆漢字ワードプロセッサの入力器として

INPUT 文を使って入力する場合，一度入力待ちになると，その間，他の仕事が出来ません．ライトペンを使って入力待ちの時間を無くしましょう．

次のプログラムでは，ライトペンを使って次々と表示される中から必要な漢字を選んで文章を作っていきます．

```
1000 DEFINT A-Z
1010 DEF FNKC(P)=(P ¥ 94)*256+P MOD 94+&H3021
1020 GOSUB *SETUP
1030 ON PEN GOSUB *PENSUB : PEN ON
1040 '
1050 *LOOP : GOTO *LOOP
1060 '
1070 *PENSUB
1080  PX=PEN(1) : PY=PEN(2)
1090  IF PY=24 THEN *MENU
1100  IF PY<0 OR 22<PY OR TMODE THEN *PENRET
```

```
1110  BEEP 1 : TMODE=-1 : SCREEN 1,0,2,4 : BEEP 0
1120  IF PX MOD 4>=2 OR PY MOD 2=1 THEN RETURN
1130  K=PAGE*240+(PY¥2)*20+PX¥4
1140  PUT(X*20,Y*25),KANJI(FNKC(K)),PSET,7,0
1150  GOSUB *MVR : GOTO *PENRET
1160 *MENU
1170  BEEP 1 : M=(PX-2)¥4 : IF M<0 THEN M=0
1180  COLOR@(M*4+2,23)-(M*4+4,23),6 : BEEP 0
1190  IF M<13 THEN PAGE=M : GOSUB *KANDISP : GOTO *PENRET
1200  ON M-12 GOSUB *SP,*MVR,*MVL,*CLSG,*BYE
1210  TMODE=-1 : SCREEN 1,0,2,4
1220 *PENRET : COLOR@(0,23)-(79,23),0
1230  PEN OFF : PEN ON : RETURN
1240 '
1250 *SP : PUT(X*20,Y*25),KANJI(&H2121),PSET,7,0
1260 *MVR : X=X+1 : IF X<32 THEN RETURN
1270  X=0 : Y=Y+1 : IF Y<7 THEN RETURN
1280  IF DBASIC THEN ROLL 25 : Y=6
               ELSE CLS 2 : X=0 : Y=0
1290  RETURN
1300 *MVL : X=X-1 : IF X>=0 THEN RETURN
1310  X=31 : Y=Y-1 : IF Y>=0 THEN RETURN
1320  X=0 : Y=0 : RETURN
1330 *CLSG : SCREEN 1,0,2,4 : CLS 2 : X=0 : Y=0 : RETURN
1340 *BYE : PEN OFF : CLS : LOCATE 0,23,1 : END
1350 '
1360 *KANDISP
1370  TMODE=0 : IF HGR THEN SCREEN 2,0 ELSE SCREEN 1,0,0,1
1380  IF DPAGE=PAGE THEN RETURN
1390  KX=0 : KY=0 : DPAGE=PAGE
1400  K1=PAGE*240 : K2=K1+239
1410  IF PAGE=12 THEN K2=2964
1420  FOR K=K1 TO K2
1430   PUT(KX,KY),KANJI(FNKC(K)),PSET,7,0
1440   KX=KX+32 : IF KX>=640 THEN KX=0 : KY=KY+KANDY
1450  NEXT
1460  IF PAGE<>12 THEN RETURN
1470  FOR K=2965 TO 3119
1480   PUT(KX,KY),KANJI(&H2121),PSET,7,0
1490   KX=KX+32 : IF KX>=640 THEN KX=0 : KY=KY+KANDY
1500  NEXT
1510  RETURN
1520 *SETUP
1530  WIDTH 80,25 : CONSOLE 0,25,0,0
1540  GOSUB *HELLOW : CLS
1550  FOR I=0 TO 17
1560   READ K$
1570   LOCATE I*4+2,23 : PRINT K$;" ";
1580   LOCATE I*4+2,24 : PRINT "███";
1590  NEXT
1600  X=0 : Y=0 : DPAGE=-1 : TMODE=-1
1610  RETURN
1620 '
1630 *HELLOW
1640  PRINT
1650  PRINT "           *** light pen ｦ ﾂｶｯﾀ ﾆｭｳﾘｮｸ ***"
1660  PRINT
1670  PRINT "ｺﾉ program ﾆﾜ light pen , ｶﾝｼﾞROM ｶﾞ ﾋﾂﾖｳﾃﾞｽ｡"
1680  PRINT
1690  PRINT "ｲﾁﾊﾞﾝ ｼﾀﾉ ｷﾞｮｳﾆ menue ｶﾞ ﾋｮｳｼﾞ ｻﾚﾏｽ｡  ";
1700  PRINT CHR$(34)"ｱｰｶ"CHR$(34)" ﾄｶ "CHR$(34)"ﾙｰﾜ"CHR$(34);
1710  PRINT " ﾄﾊ ｶﾝｼﾞ ﾉ ﾖﾐ ﾉ ｺﾄﾃﾞｽ｡"
1720  PRINT
1730  PRINT "ﾏﾀ "CHR$(34)"   "CHR$(34)" ﾄｶ "CHR$(34)"-->"CHR$(34);
1740  PRINT " "CHR$(34)"<--"CHR$(34)
           " ﾊ space move-left move-right ﾉ ｺﾄﾃﾞｽ｡"
1750  PRINT
1760  SCREEN 0,3 : CLS 2 : SCREEN 1,0,2,4
1770  PRINT " ｺｳﾌﾞﾝｶｲﾉｳ display ﾃﾞｽｶ (y/n) "; : ANS$=INPUT$(1)
1780  IF ANS$="y" OR ANS$="Y" THEN PRINT "y" : HGR=-1 : KANDY=32
                               ELSE PRINT "n" : HGR=0 : KANDY=16
1790  PRINT : PRINT " Disk Basic ﾃﾞｽｶ (y/n) "; : ANS$=INPUT$(1)
1800  IF ANS$="y" OR ANS$="Y" THEN PRINT "y" : DBASIC=-1
                               ELSE PRINT "n" : DBASIC=0
```

```
1810  LOCATE ,,0 : FOR DELAY=0 TO 1000 : NEXT
1820  RETURN
1830 '
1840 DATA ｱｰｶ,ｶｰｷ,ｷｰｸ,ｸｰｻ,ｻｰｼ,ｼｰｼ,ｼｰｿ,ｿｰﾂ,ﾂｰﾆ,ﾆｰﾋ,ﾋｰﾐ,ﾐｰﾙ,ﾙｰﾜ
1850 DATA "    ",-->,<--,clr,end
```

☆位置の読み取り

ライトペンを使うと，画面上の任意の位置を指定できますが，これはライトペンの最も基本的な使い方です．

次のプログラムを実行させると，ライトペンをチョーク代わりにして画面上に任意の図形が描けます．

```
100 '
110 ' light pen demo program
120 '
130 ON KEY GOSUB *NEXTCOLOR,*CLEARCRT,*TERMINATE
140 KEY ON
150 SCREEN 0,3 : COLOR 7,7,0,7 : CLS 2 : COL=7
160 CONSOLE 0,25,0,1 : WIDTH 80,25
170 PRINT
180 PRINT "ﾗｲﾄﾍﾟﾝ ｦ ﾂｶｯﾀ ｵｴｶｷ program"
190 PRINT
200 PRINT "  ｲﾛｦ ｶｴﾙﾄｷ ﾊ Key-1 ｦ"
210 PRINT "  ｶﾞﾒﾝ ｦ ｹｽﾄｷ ﾊ key-2 ｦ"
220 PRINT "  ﾔﾒﾙﾄｷ ﾊ  key-3 ｦ ｵｼﾃ ｸﾀﾞｻｲ｡"
230 PRINT
240 PRINT " ﾅﾆｶ ﾃｷﾄｳﾅ ｷｰｦ ｵｽﾄ ﾊｼﾞﾏﾘﾏｽ｡"
250 DUMY$=INPUT$(1) : CLS : SCREEN ,0
260 PEN ON
270 *PENWAIT
280  IF PEN(0)=0 THEN *PENWAIT
290 *LOOP
300  LOCATE PEN(1),PEN(2),0 : PRINT "■";
310  GOTO *LOOP
320 *NEXTCOLOR : COL=(COL+1) MOD 8 : COLOR COL : RETURN
330 *CLEARCRT : CLS : RETURN
340 *TERMINATE
350  COLOR 7 : KEY OFF : SCREEN ,3 : LOCATE ,,1
360  END
```

7章　うまいプログラミングのために

ここまで本書を読んできたみなさんは，ちょっとしたプログラムなら簡単に作れるようになったと思います．しかし，長いプログラムを作ると，プログラムがエラーだらけになって困ることがあるでしょう．ここで言うエラーとは，単にBASICが「誤りですよ」と表示してくれるエラーだけでなく，正しい結果が得られなかったり，みなさんが考えた通りに動作しなかったりすることも含んでいます．これらのエラーをすべて取り除いて完全に動くプログラムにしないと，そのプログラムを作った意味がなくなってしまいます．そこでここでは，「うまいプログラムとは何か」ということを考えてみましょう．

7.1　うまいプログラミングとは

ちょっとしたプログラムが作れるようになると，次に問題となるのは「何のプログラムを作るか」や「どんな風にプログラムを作るか」ということでしょう．前者はコンピュータにさせる仕事の内容の事で，後者はプログラミングテクニックの事で，どちらもでき上ったプログラムの質を左右する条件になるでしょう．

では，うまいプログラムとはどういうものでしょうか．うまいプログラムとは，役に立つプログラムのことを言うのでしょうか．それなら給与計算などのビジネスプログラムでしょうか．しかし，ある意味では，ゲームプログラムも役に立ちます．つまり，役に立つプログラムであると決めるのは，最終的にはそのプログラムを使う人ということになりそうです．

この章では，〝うまいプログラムとは何か〟という事を考えてみたいと思います．

まず，プログラムのできばえを判断する要因として，先程述べた2つがあります．1つは，「何のプログラムを作るか」という仕事の内容であり，もう1つは，それを具体化する「プログラムテクニック」です．前者は，あくまでプログラマ自身が考える事柄であり，本書の意とする処ではありませんし，また人から教わるものでもないでしょう．問題は後者です．コンピュータに実行させる仕事の内容を，コンピュータが提供するいくつかの命令の中で，具現化していかねばなりません．ここでは，この問題に主眼をおき，うまいプログラムを作るためのテクニックを，い

くつかの条件に分け，考えていきたいと思います．

幸い，N_{88}-BASICは，私たちに多くの便利な命令を提供してくれています．それらを使わない手はありません．ここでは，「うまいプログラム作り」を通して，今まで説明しきれなかったN_{88}-BASICの命令の解説もいっしょにしていくつもりです．

7.1.1 うまいプログラムの条件

① 完全に動作する

「こんなことは当り前だ，わざわざ言う必要はないじゃないか」と考えることでしょう．でも，何百行から何千行にも渡る長いプログラムになると，完全に動作するようにさせるのは，至難なことです．この程度の長さのプログラムになると，いろいろな条件の下でテストをくり返して，完全にバグ（エラーのこと）を取り除いたと思っても，一般に，8割から9割程度しかバグを取り除いていないと言われます．プログラムが完成してから1年たってバグが見つかったというのはざらにあります．

プログラムというのは，どんな場合に遭遇しても，正しく動作しなければなりません．正しい動作というのは，エラーがないということだけでなく，目的に沿う動作をして正しい結果が得られることです．特に間違ったデータを入力した時に，エラーを起こさないだけでなく，「今，入力したデータは誤っていますよ」と表示するぐらいのことが必要です．

② 動作・実行が速い

速さの追求というのは，コンピュータの宿命とも言えるもので，現在の大型コンピュータの開発の大きな目標となっています．本書が対称としているPC-8801は，残念ながらハードウェアの改善はほとんど無理ですので，ソフトウェア（プログラムのこと）の改善によって速度の向上を目指しましょう．

それにはプログラムのアルゴリズム（解法手順）をしっかり考える必要があります．よく考えられたアルゴリズムのプログラムは，すばらしい速さを見せてくれます．

③ プログラムが短い

プログラムが短いということは，使用するメモリの量が少なくて済むということです．このことは特に多量のデータを扱う時に有利になります．また，プログラムが短いということは，一般に動作が速いということにも関係しているようです．プログラムを管理する上でもプログラムが短い方がいいですし，同じ仕事をするプログラムでは短い方が良いでしょう．でも，へたに短くするとプログラムが分かりにくくなって，デバッグがしずらくなります．

④　プログラムが分かり易い

皆さんは他人の作ったプログラムを見たことがありますか．また自分の作ったプログラムを長い時間一例えば数ケ月たって見直した時，そのプログラムが何のためのプログラムで，どんなアルゴリズムになっているか分かりますか．もし，ちんぷんかんぷんなプログラムなら，もう二度と見たくなくなるでしょう．後々のメンテナンスや使用のために，分かりやすいプログラムということは大切な事なのです．

でも残念ながら，分かり易いプログラムを作る時は，②動作・実行が速い．とか．③プログラムが短い．ということを，少しは犠性にしなければなりません．背に腹は代えられませんから．

⑤　プログラムが使い易い

皆さんが作ったプログラムは自分だけで使いますか？　もし，他の人にも使ってもらう様なことがあるのでしたら，使い易さは重要です．もちろん，自分で使う時も使い易さは重要です．使いにくいプログラムは良いプログラムとはいえません．

⑥　プログラムの資料が備っている

プログラムには，大抵マニュアルや仕様書のような物が付いているものです．プロは，プログラムを作るのにかけた時間以上の時間を，マニュアルや仕様書を作るのにかけます．私たちの作るプログラムに仕様書やマニュアルなどが無くてもよいという理由はありません．他の人に使える様に，また保守のためにも，そして，④プログラムが分かり易い．ためや，⑤プログラムが使い易いためにも，仕様書やマニュアルはしっかりと作っておきたいものです．

いろいろと述べてきましたが，これだけがうまいプログラムの条件ではないでしょう．あなたにはどんなことが思い浮かびますか．ここで述べたことは文中でも述べた通り，互いに反する事や関連する事がありますが，いずれも重要ですので，しっかりと頭に入れておいてください．

しかし，やはり一番の重要事項は，

「そのプログラムを使うメリットがある」

ことでしょう．仕事の量が減ることや，実行時間の短縮などのメリットを期待してプログラムを作ったのに，逆に仕事のわずらわしさや量を増やすというデメリットが増えたのでは，そのプログラムの存在する理由がなくなってしまいます．もちろんわざわざこのようなプログラムを作る人はいませんが，使用する目的を的確に摑んでいなかったり，考えが浅かったりすると．往々にして能率が悪いプログラムを作ってしまうことがあります．

でも，最初から分かり易く，使い易い，便利なプログラムが作れるはずはありません．初めは，無理，無駄の多いプログラムしかできないでしょう．しかし，実際にプログラムを作ってみることは大切です．そして，自分の作ったプログラムを見直して，一歩一歩改良していくのです．重要なことは自分のプログラムを反省してみることです．決して作り放しにしないでください．

また，他人のプログラムを解析・理解して活用することも大切です．独力では浮かばなかったアルゴリズムを覚えることもできますし，また，他人のプログラムをヒントにして，別のアルゴリズムが浮かぶこともあります．

とにかく，より良いプログラムを作ろうとする向上心を，常に持っていることが必要です．

7.2 実行速度向上のために

BASICの実行速度はそんなに遅いものではありませんが，速度に不満を持っている人がいると思います．そして，BASICのプログラムは，いろいろと手を入れてもそんなに速くならないように思われがちですが，実は違います．BASICはプログラムの最適化（コンピュータ内部で速度を向上させようとするなどの働き）を行いませんので，実行速度の向上をめざすには，私たちがプログラムそのものを改良する必要があります．ここでお話しすることは，そのためのヒントと考えてください．

7.2.1 アルゴリズムを見直す

プログラムの長さがもっと短くならないか，ループの回数を減らせないかを考えてみましょう．例えば，ソートのプログラムでは，より実行の速いアルゴリズム（クイックソートなど）を使ってみましょう．

また，プログラムを一所懸命，編集していると，GOTO文が多くなってしまうことがあります．1つでもGOTO文を減らしましょう．図の左側のプログラムは（ロ）の部分と（ハ）の部分が入れかわっているため，右側のプログラムに比べて，2つのGOTO文が必要になっています．こんな無駄なGOTO文は省きましょう．無駄を省いただけ実行速度の向上が見られます．

（イ）
GOTO
（ハ）
（ロ）
GOTO
無駄なGOTO文がある

無駄なGOTO文を省いて，
このように改善しよう．

（イ）
（ロ）
（ハ）
すっきりとしたプログラム

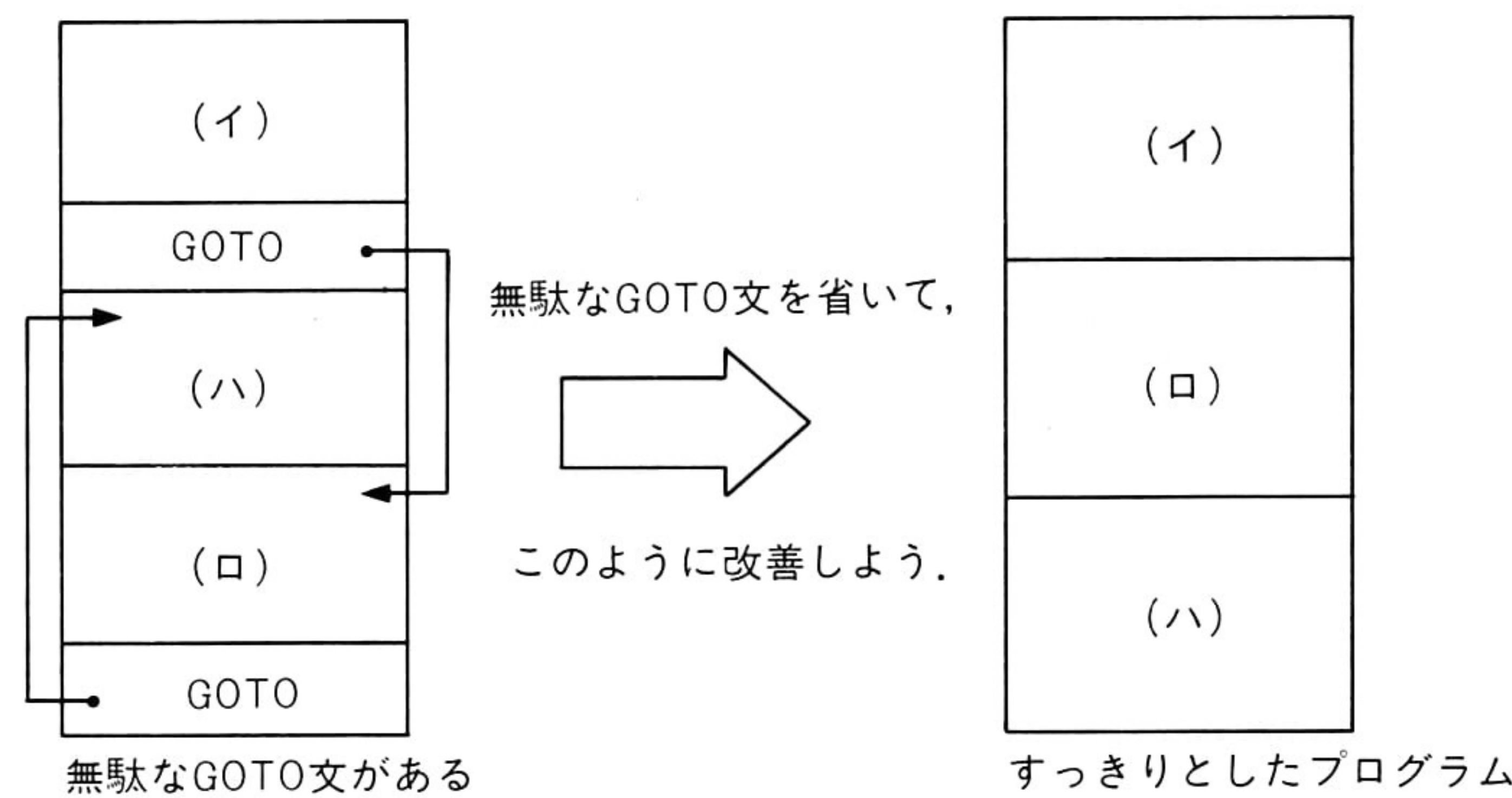



図1　無駄なGOTO文を省く

もう一つGOTO文についての提言を与えましょう．GOTO文やGOSUB文の行き先などの行番号は，プログラムの最初から探します．そのため良く使うサブルーチンはプログラムの最初

に持って行った方が良いのです．

GOTO

最も使用頻度の高いサブルーチン

最も使用頻度の低いサブルーチン

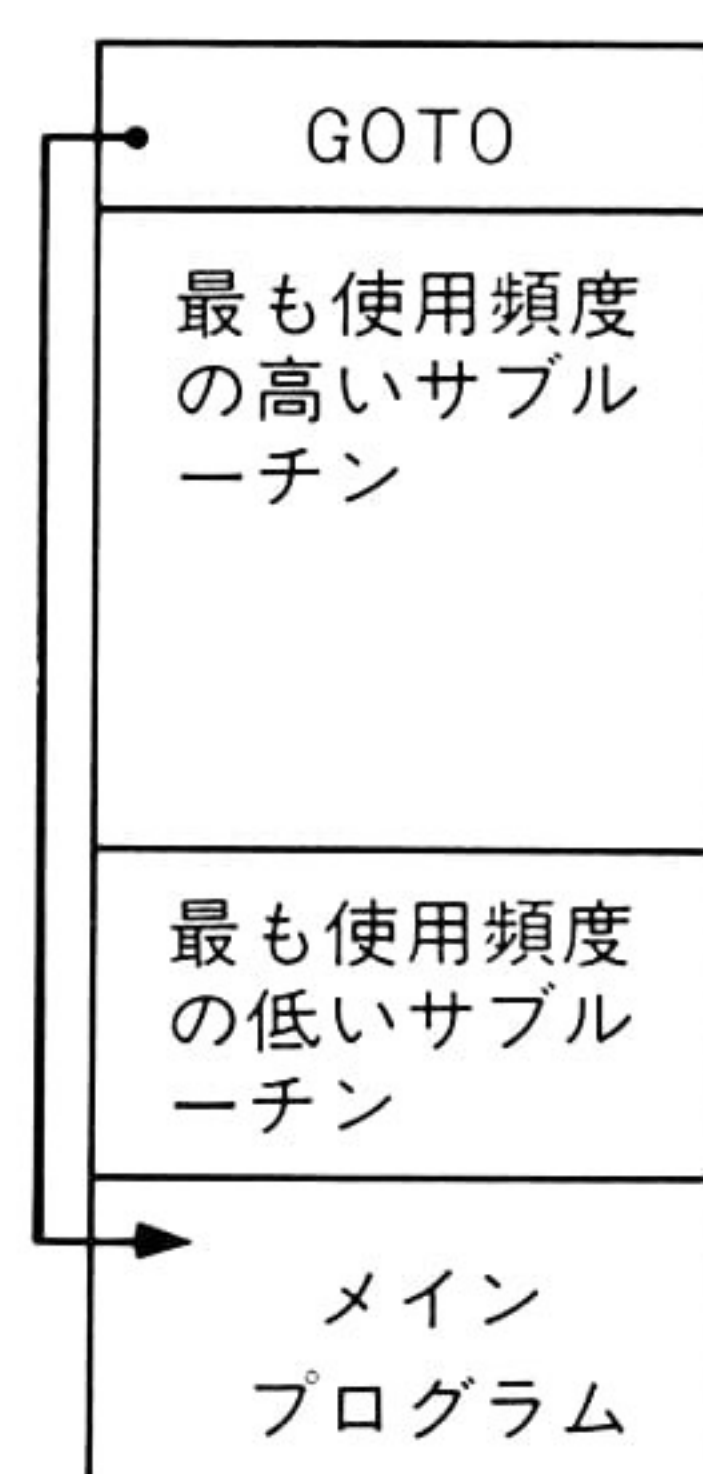
メインプログラム

行番号で飛び先を指定する時は，実行速度をあげるため左のようにするとよい．

図2　行番号で飛び先を指定する場合のプログラムの例)

ただし，これは飛び先に行番号を与えた場合の注意で，ラベル名を使って飛び先を指定する場合はこれと異なります．

プログラムを RUN で実行させると，最初にプログラム中で使われているラベルを全て探し出し，そのラベルとラベルの宣言されている位置との対応表を作り出します．このためラベルを使うと飛び先をプログラム中から探し出す事無く，対応表より直接飛び先の位置を求めますので，サブルーチンの配置はどの様にしても実行速度はほとんど変化しません．ですからなるべく，GOTO 文や GOSUB 文の飛び先は，ラベル名を使って指定する事をお勧めします．

FOR ～ NEXT 文や WHILE ～ WEND 文を使えるのに，IF ～ THEN 文を使ってループを作っているのも，よくあることです．

IF ～ THEN 文を使うと GOTO 文の場合と同様に飛び先を探し出さなければなりませんし，またラベル名を使っても対応表から位置を求めるのに，わずかですが時間を費やします．しかし，FOR ～ NEXT 文や WHILE ～ WEND 文を使うと，飛び先をプログラム中からサーチする時間が減りますし，ラベル名を付けるのに頭を悩ます必要もありません．

ですからループを作る場合はなるべく FOR ～ NEXT 文や WHILE ～ WEND 文を使って下さい．

また FOR ～ NEXT ループで，NEXT 文に変数名を与えると対応する FOR 文を探し出すため時間がかかりますので，NEXT 文で制御変数が明らかな場合には，これを省略した方が良い結果が得られます．(WHILE ～ WEND は，DISK BASIC でのみサポートされています)．

7.2.2　余分なプログラムを取り除く

例えば REM 文などは実行は行いませんが，読み飛ばすことに時間がかかります．ですから REM 文は取り除いた方が実行速度が上がりますが，これでは REM 文を使う意義がなくなり，プログラムが理解しにくくなります．REM 文は実行速度を最優先する時にだけ取り除く様にして下さい．

7.2.3 プログラムを短くする

まず思い付くのは，プログラム中の余分な空白を取り除く事です．

```
100 ' ハイレツ ノ クリヤ
110 DIM A(10,10,10)
120 FOR I=0 TO 10
130  FOR J=0 TO 10
140   FOR K=0 TO 10
150    A(I,J,K)=0
160   NEXT
170  NEXT
180 NEXT
190 END
```

このプログラムは，実行に影響を与えない空白が多くありますので，これを取り除く事で実行速度を上げる事が出来ます．

```
100 ' ハイレツ ノ クリヤ
110 DIM A(10,10,10)
120 FOR I=0 TO 10
130 FOR J=0 TO 10
140 FOR K=0 TO 10
150 A(I,J,K)=0
160 NEXT
170 NEXT
180 NEXT
190 END
```

しかし，これによってプログラムの構造を見づらくしています．最初のリストではループの流れを明らかにする様，各行の始まりに空白を置いて段付けをしています．これはインデンテーションと呼ばれるプログラムを見やすくする一般的方法ですので，この変更はあまりお勧めできません．

またステートメント間の空白を取り除く事はミスの元です．次の例では空白を取ったために，FOR 文が代入文に化けてしまっています．

```
FOR I = A TO B STEP C
          ⇩
FORI = ATOBSTEPC
```

1行に文をいくつも書くマルチ・ステートメントによってもプログラムを短くする事が出来ます．しかしこれも REM 文と同様，あまり1行に多く詰め込みますと，プログラムは見ずらくなります．

7.2.4 頻繁に使う変数は始めに定義する

変数はプログラム実行中に現われた順にその領域が確保されて，探索はその順に行われますので，頻度の高い変数は初めに代入を行った方が実行速度が向上します．

```
100 BEGIN=0:LIMIT=10000
110 FOR I=BEGIN TO LIMIT
120   SUM=SUM+1
130 NEXT
140 PRINT SUM
150 END
```

```
100 SUM=0:I=0:LIMIT=10000:BEGIN=0
110 FOR I=BEGIN TO LIMIT
120   SUM=SUM+1
130 NEXT
140 PRINT SUM
150 END
```

7.2.5 変数はなるべく整数型を使う

また，配列変数はすべての単純変数の後にその領域が確保されますので，新しい単純変数を使う度に配列変数がメモリ上を動きます．その影響は，配列変数が大きければ大きいほど実行時間の増大として現れます．この影響は，事の他大きいもので，何千個という配列を確保した後，新しい単純変数を作ると，その時間は目に見える程です．

大きな配列を確保する時などは，必ず配列変数定義の前に，単純変数を定義し（初期値を代入しておく）して下さい．

BASICが使えるメモリ	
プログラム	行番号の小さい順に並んでいる．
単純変数	参照した順に並んでいる．
配列変数	参照した順に並んでいる．
未使用	

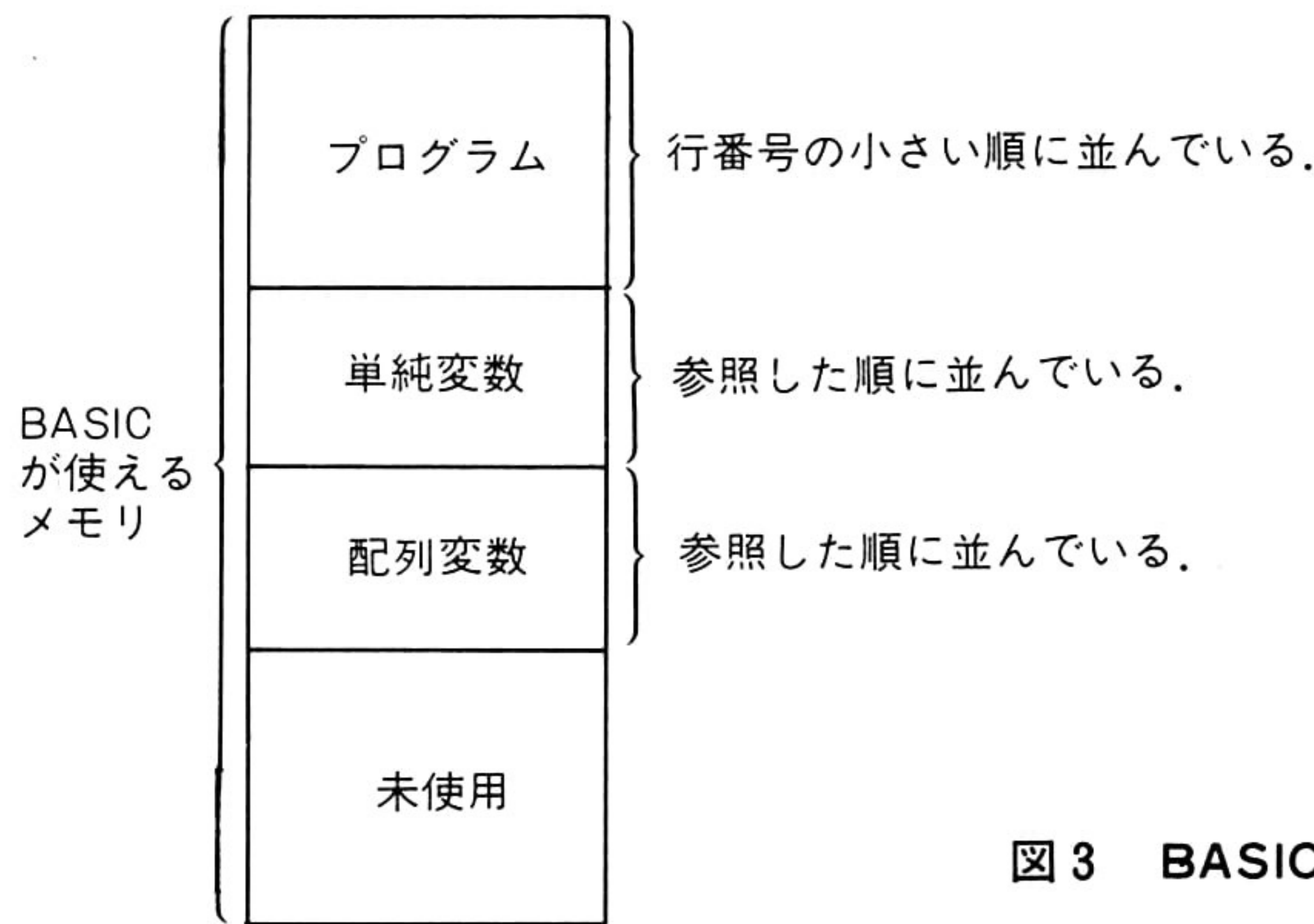
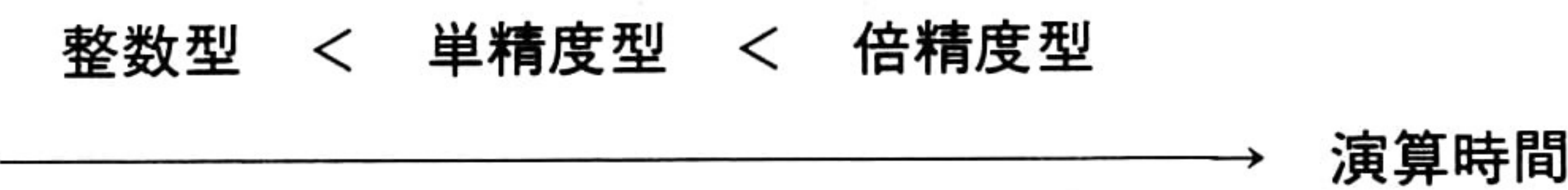



図3　BASICのワークエリアのメモリ配置

数値の演算時間はその型により異なります．

整数型　<　単精度型　<　倍精度型

———————————————→ **演算時間**

このため，なるべく計算時間の短い型を使った方が有利です．変数を使う場合，型指定を省略すると単精度型になります．そこで扱う数値が整数の場合は，DEF INT を実行するか，または変数名に％を付けるようにしましょう．

この様に，実行速度の向上にはいくつか方法がありますが，最も重要なのはアルゴリズムです．

良く考えられたアルゴリズムは，多少プログラムが複雑になっても，それを無視できるほどの実行速度の向上を得る事が出来ます.

ここでは，その例として，アルゴリズムが実行速度に大きく影響するソーティング・プログラム（値が順番になる様に並び変えるプログラム）を3種類あげてその実行速度を比較してみます.

```
1000 '
1010 ' sorting sample  by cat
1020 '
1030  DEFINT A-Z
1040  N=100 : M=N
1050  I=0 : J=0 : T$=""
1060  W=0 : L=0 : R=0 : SP=0
1070  HH!=0 : MM=0 : SS=0
1080  DIM SPR(M),SPL(M),S(N),D(N)
1090  FOR I=1 TO N : D(I)=RND*1000: NEXT
1100  PRINT "number of data";N
1110  GOSUB *SETUP1
1120  GOSUB *DISPLAY
1130 '
1140  GOSUB *SETUP
1150  PRINT "ﾊﾞﾌﾞﾙ sort"
1160  GOSUB *SORT1
1170  GOSUB *RESULT
1180 '
1190  GOSUB *SETUP
1200  PRINT "ﾀﾝｼﾞｭﾝ ｿｳﾆｭｳ sort"
1210  GOSUB *SORT2
1220  GOSUB *RESULT
1230 '
1240  GOSUB *SETUP
1250  PRINT "Quick sort"
1260  GOSUB *SORT3
1270  GOSUB *RESULT
1280 '
1290  END
1300 '
1310 *SETUP
1320  GOSUB *TIME : STIME!=TIME!
1330 *SETUP1
1340  FOR I=1 TO N : S(I)=D(I) : NEXT
1350  RETURN
1360 '
1370 *SORT1
1380  FOR I=2 TO N
1390   FOR J=N TO I STEP -1
1400    IF S(J-1)>S(J) THEN SWAP S(J-1),S(J)
1410   NEXT
1420  NEXT
1430  RETURN
1440 '
1450 *SORT2
1460  FOR I=2 TO N
1470   W=S(I) : S(0)=W : J=I-1
1480  *SORT21
1490   IF W<S(J) THEN S(J+1)=S(J) : J=J-1 : GOTO *SORT21
1500   S(J+1)=W
1510  NEXT
1520 RETURN
1530 '
1540 *SORT3
1550 SP=0 : L=1 : R=N : GOTO *SORT31
1560 *SORT30 : R=SPR(SP) : L=SPL(SP) : SP=SP-1
1570  *SORT31 : I=L : J=R : X= S((L+R)¥2)
1580   *SORT32
1590     IF S(I)>=X THEN *EXIT1
1600     I=I+1 : GOTO *SORT32
1610    *EXIT1 : IF X>=S(J) THEN *EXIT2
1620     J=J-1 : GOTO *EXIT1
1630    *EXIT2 : IF I=<J THEN SWAP S(I),S(J):I=I+1:J=J-1:GOTO *SORT32
1640    IF J-L<=R-I THEN *SORT33
```

```
1650     IF I<R THEN SP=SP+1 : SPL(SP)=I : SPR(SP)=R
1660     R=J : GOTO *SORT34
1670    *SORT33
1680     IF L<J THEN SP=SP+1 : SPL(SP)=L : SPR(SP)=J
1690     L=I
1700    *SORT34
1710   IF L<R THEN *SORT31
1720  IF SP<>0 THEN *SORT30
1730 RETURN
1740 '
1750 *RESULT
1760  GOSUB *TIME : ETIME!=TIME!
1770  TIME!=ETIME!-STIME!
1780  PRINT "used time:";TIME!;"seconds"
1790  GOSUB *DISPLAY
1800 RETURN
1810 '
1820 *TIME
1830  T$=TIME$
1840  HH!=CSNG(VAL(LEFT$(T$,2))*3600)
1850  MM=CSNG(VAL(MID$(T$,4,2))*60)
1860  SS=CSNG(VAL(RIGHT$(T$,2)))
1870  TIME!=HH!+MM+SS
1880 RETURN
1890 '
1900 *DISPLAY
1910  FOR I=1 TO N
1920   PRINT USING " ###";S(I);
1930   IF I MOD 18=0 THEN PRINT
1940  NEXT
1950  PRINT
1960 RETURN


run
number of data 100
 245 305 312 515  58 789 497 364 985 902 727   7 969   2 956  41 897 660
 554 819 907 858 869 507 584 448 868  33 604 779 287 785 137 227 215 877
 857 568 365  33 876 763 201 609 374 226 746 257 930 453  40 885   3  43
 278 495 749 377 733 127 379 879  21 201 696  43 579 711 892 229 130 305
 117 705 789 549 288 806 482 993 134 862 126  97 767 429 763 982 803 993
 154 629  35 433 219 540  19   4 485 954
ﾊﾞﾌﾞﾙ sort
used time: 54 seconds
   2   3   4   7  19  21  33  33  35  40  41  43  43  58  97 117 126 127
 130 134 137 154 201 201 215 219 226 227 229 245 257 278 287 288 305 305
 312 364 365 374 377 379 429 433 448 453 482 485 495 497 507 515 540 549
 554 568 579 584 604 609 629 660 696 705 711 727 733 746 749 763 763 767
 779 785 789 789 803 806 819 857 858 862 868 869 876 877 879 885 892 897
 902 907 930 954 956 969 982 985 993 993
ﾀﾝｼﾞｭﾝ ｿｳﾆｭｳ sort
used time: 40 seconds
   2   3   4   7  19  21  33  33  35  40  41  43  43  58  97 117 126 127
 130 134 137 154 201 201 215 219 226 227 229 245 257 278 287 288 305 305
 312 364 365 374 377 379 429 433 448 453 482 485 495 497 507 515 540 549
 554 568 579 584 604 609 629 660 696 705 711 727 733 746 749 763 763 767
 779 785 789 789 803 806 819 857 858 862 868 869 876 877 879 885 892 897
 902 907 930 954 956 969 982 985 993 993
Quick sort
used time: 16 seconds
   2   3   4   7  19  21  33  33  35  40  41  43  43  58  97 117 126 127
 130 134 137 154 201 201 215 219 226 227 229 245 257 278 287 288 305 305
 312 364 365 374 377 379 429 433 448 453 482 485 495 497 507 515 540 549
 554 568 579 584 604 609 629 660 696 705 711 727 733 746 749 763 763 767
 779 785 789 789 803 806 819 857 858 862 868 869 876 877 879 885 892 897
 902 907 930 954 956 969 982 985 993 993
Ok
█
```

7.3 プログラムを分かり易くするためには

プログラムが分かり易い事は多くのメリットがあります.

① デバッグ時間が短縮できる.

② プログラムの仕様を変更する際に, それが簡単になる.

③ 他人が読んだ時や, 自分で後になって読み返す時にプログラムの流れをつかむのが容易になる.

④ プログラムのメンテナンスに役立つ.

これらの利点は, プログラムの作成時には感じないかもしれませんが, デバッグ時や後にプログラムをさらに改良する際に, 大きく現れます. そして, ひいてはあなた自身が, プログラムを良く理解する事で, プログラミング技術が向上する事にもつながります. ですから, プログラム作成時にはこの「分かりやすいプログラムを作る」事を忘れないで下さい.

では, どの様にしてプログラムを分かり易くするかについて説明しましょう.

1. 変数名, ラベル名は意味のある名前を付ける

この事で変数に代入されている値の意味や, 呼び出しているサブルーチンの意味が明確となります.

2. プログラムは段付けを行う

FOR ~ NEXT や WHILE ~ WEND ループにおいては, ループ内を一段下げたり, ラベル名や行番号で指定されている行の次から一段下げる事で, プログラムの流れが明らかになります.

3. プログラムをモジュール化する

プログラム作成において, いきなり全部のプログラムを作ろうとせずに, まず全ての処理を, 大まかな内容でいくつかのモジュールに分割します. そして, さらにその各々のモジュール内を分けていく事を繰り返すのです. こうする事でプログラムは, 小さなプログラムの集まりとなります. これによりプログラムの構造が分かり易くなります. 一般的にメインルーチンは, 短い方がプログラムは理解し易い様です.

また, この方法でプログラムを作成した時には, デバックは, 1つのモジュールごとに行い, バグを追いつめていくと, デバック時間を短縮する事が出来ます.

```
GOSUB 〈イニシャライズルーチン〉
WHILE
```

```
  GOSUB 〈入力ルーチン〉
  GOSUB 〈内部処理ルーチン〉      メインループ
  GOSUB 〈出力ルーチン〉
WEND
GOSUB 〈終了処理ルーチン〉
END
```

4. REM 文を使ってプログラム中にコメントを入れる

プログラムを作成している時は，この変数の値は何を表すかとか，使っているアルゴリズムやサブルーチン内で使用する変数などは理解していますが，ひとたびそのプログラムから遠ざかっていて，しかもそれについての記録を書き止めていないと，プログラムの内容を忘れてしまいます．プログラム作成を，詳細な仕様書を作りながら進めればこの問題は解決しますが，これでは大変ですね．そこでこれらの情報をREM文を使ってコメントとしてプログラムに含ませるのです．この事はプログラムの理解度を高めるのに有効な方法で，多少なりともコメントがあると，大きく違います．

7.4 使い易いプログラムを作るために―入力編―

プログラムの使い易さは，データの入力の仕方や，結果の表示が美しく見易い事と密接に関係します．これはプログラムを使うのは人間だ，と言う事を考えれば至極当然なことです．そこでまず，入力方法について検討しましょう．

数値や文字列の入力方法としてこれまでINPUT文を使ってきましたが，BASICには入力文として，他にも幾つかの文や関数が用意されていますので，ここで紹介してみましょう．

1. LINE INPUT

INPUT文で文字列を入力することができましたが，コンマ（，）を入力する事は出来ませんでした．この場合はクォーテーションマーク（"）で囲まなければなりませんでした．また，クォーテーションマーク自身も入力できませんでしたね．そこで，こんな不便さを解消する文として，LINE INPUT文があります．

```
LINE INPUT  [〈プロンプト文〉]；〈文字変数〉
```

LINE INPUT文は，入力行全体（キャリッジ・リターンまで）を1行とし読み込むので，コンマやクォーテーションマークを入力する事が出来ます．またこの文は，実行しても入力促進文字であるクエッションマークは表示しませんので，表示したい場合にはプロンプト文中に？を入れておきます．

```
list
100 LINE INPUT IN$
110 PRINT IN$
120 LINE INPUT "? ";IN$
130 PRINT IN$
140 END
Ok
run
a,b,c,d,e,f
a,b,c,d,e,f
? A,B,C,D,E,F
A,B,C,D,E,F
Ok
■
```

2. INPUT WAIT

INPUT 文を実行すると入力待ちとなり，リターンキーを入力するまでプログラムの実行は停止します．しかし，ある時間内に入力されなかったら入力待ちをやめて次の処理へ進みたい場合がありますね．そこで入力待ち時間を制限する文として，INPUT WAIT 文があります．

INPUT WAIT　〈待ち時間〉，[〈プロンプト文〉；]〈変数名〉[，〈変数名〉……]

待ち時間は0.1秒単位で指定しますので例えば，

INPUT WAIT　50，X

とすれば待ち時間は5秒となります．

また，この文の後にマルチステートメントとして他の文を続けた場合，時間内に入力が行われた時のみ後の文を実行し，時間切れの時は後の行を無視して次の行を実行します．そこで一般にこの文を使う場合は次の例の様にマルチステートメントを使います．

```
100 *LOOP
110  INPUT WAIT 40,X          : GOTO *SKIP
120  PRINT "ﾊﾖ ﾆｭｳﾘｮｸ ｾﾝｶ!"  : GOTO *LOOP
130 *SKIP
140  PRINT X
150 END
```

また，時間切れの場合，入力に指定した変数の値は変化しません．

3. LINE INPUT WAIT

この文は見て分かるように，LINE INPUT 文と INPUT WAIT 文を1つの文にしたものですから詳しくはそちらを見て下さい．

LINE INPUT WAIT　〈待ち時間〉，[〈プロンプト文〉；]〈文字変数〉

4. INKEY$

この関数は，キーが押されている時はその文字を，押されていない時はスルストリングを返し

ます．したがって，この関数は入力待ちを行わずに，リアルタイムにキーボードから入力できます．

```
100 *LOOP
110  K$=INKEY$
120  IF K$="" THEN K$="."
130  PRINT K$;
140  GOTO *LOOP
```

また，この関数はリターンキーやタブキーなどコントロールキャラクタも入力できますが，例外として次のキーは入力できません．

STOP キー，CTRL-C，CTRL-S，COPY キー．

さて，先程のプログラムを変更してゆっくり表示するようにしてみます．

```
100 *LOOP
110  K$=INKEY$
120  IF K$="" THEN K$="."
130  FOR DELAY=0 TO 50 : NEXT
140  PRINT K$;
150  GOTO *LOOP
```

キーを素早く押したりした場合，キーから手を放した後も表示されます．これは N_{88}-BASIC がキーボード用の入力バッファを持っていて，キーを押すと入力文以外の時でもそれを読み取ってバッファにためています．INKEY$ は，そのバッファから文字を読む事を示しているのです．ですから INKEY$ は正確には，キー入力バッファの先頭の文字，バッファが空の時はヌルストリング，を返す関数なのです．

ただし，STOP キーなどでプログラムを中断した場合にはバッファの内容はクリアされます．

5. INPUT$

S$ = INPUT$ (<文字数>)

これは，文字数を指定してキーボードより入力する関数です．この関数は，INKEY$ と違い指定した文字数が入力されるまで入力待ちを行いますが，入力できる文字は STOP キー，CTRL-C，COPY キー，CTRL-O 以外全てです．

また，この関数を使って入力した時は，INKEY$ と同様に押したキーは表示されません．

```
100 PRINT "input$ / sample program"
110 PRINT : PRINT "ﾃｷﾄｳﾅ ｷｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
120 PRINT "return ｷｰ ｦ ｵｽﾄ ﾆｭｳﾘｮｸｼﾀ ﾓｼﾞﾚﾂ ｶﾞ ﾋｮｳｼﾞ ｻﾚﾏｽ"
130 L$=""
140 *LOOP
150  I$=INPUT$(1)
160  IF I$=CHR$(13) THEN PRINT L$ : L$="" ELSE L$=L$+I$
170  GOTO *LOOP
```

6. 入力文の変わった使い方

入力文は，単に数値や文字を入力する以外に，プログラムの実行を外部と同期を取る用途にも使います.

例えば，画面への表示が画面に納まらない時，ただ表示していたのでは，次々とスクロールしてしまいますね. そこで，CTRL-S を押してプログラムを止めたりしますが，これを１画面分表示したら入力待ちにさせ，何かキーを押す事で次の１画面分の表示をするのはどうでしょう.

良く使われる例としてこんな INPUT$ の使い方があります.

```
DUMMY$ = INPUT$ (1)
```

また数値を入力させる時に，INPUT 文で数値変数を指定するとは限りません.

いったん文字列として入力し，次に VAL 関数を使って数値に変換する事も良くあります.

例えば,数値を入力する際にその前に &H や&O を付けると16進表記や８進表記が使えますが,仮に入力はすべて16進表記とした場合にはいちいち &H と打つのは面倒ですね. そこで次の様な入力方法もあります.

```
100 *LOOP
110  PRINT "16 ｼﾝｽｳ ";
120  INPUT H$
130  H=VAL("&H"+H$)
140  PRINT " ｦ 10 ｼﾝｽｳ ﾆ ｽﾙﾄ ";H;"ﾄ ﾅﾘﾏｽ｡"
150  GOTO *LOOP
```

また，キーを打つと通常は画面に表示（エコーバック）しますが，INPUT$，INKEY$ では表示されませんね. そこでこんなのはどうでしょう.

プログラムを実行させるにはまずパスワードを入力し，それが正しくないと実行できなくする. そしてこの時に入力したパスワードを他人に見られないために，押したキーに対応した文字の画面への表示を行わない様にする.

ちょっとわざとらしい例ですが，ゲーム等で応用する事もできますね.

7. 入力をし易くするために

入力文を使う場合，単にクエッションマークが表示されたので，数値や文字を入力する，というのではあまりに味気ないと思いませんか？せめて「何々の値を入力して下さい」という表示はあってしかるべきと思いませんか？ いくらこのプログラムは自分しか使わないからと言っても，クエッションマークだけではあまりに芸がなさすぎます. そこで PRINT 文や，INPUT 文のプロンプト文を使って入力する値の意味を表示すると良いでしょう.

また，キー割り込みの所でも述べましたが，入力待ちの時に HELP キーが押されると入力方法の説明が表示されるのはどうでしょう. 次のプログラムは HELP キーで，何を入力すれば良いかが表示されます.

```
100 '
110 *LOOP
120  ON HELP GOSUB *HELP1 : HELP ON
130  FLAG=0 : INPUT "begin time (HH:MM:SS) ";TB$
140  IF FLAG THEN 120
150  ON HELP GOSUB *HELP2 : HELP ON
160  FLAG=0 : INPUT "end time (HH:MM:SS) ";TE$
170  IF FLAG THEN 150
180  H1=VAL(LEFT$(TE$,2)) : H0=VAL(LEFT$(TB$,2))
190  M1=VAL(MID$(TE$,4,2)) : M0=VAL(MID$(TB$,4,2))
200  S1=VAL(MID$(TE$,7,2)) : S0=VAL(MID$(TB$,7,2))
210  S=S1-S0 : IF S<0 THEN S=S+60 : M1=M1-1
220  M=M1-M0 : IF M<0 THEN M=M+60 : H1=H1-1
230  H=H1-H0 : IF H<0 THEN H=H+24
240  TOTAL=(H*60+M)*60+S
250  PRINT TB$"-->"TE$" == "TOTAL" 秒"
260  GOTO *LOOP
270 '
280 *HELP1
290  PRINT
300  PRINT "ハジメ ノ ジコク ヲ ニュウリョク シテクダサイ。"
310  GOTO *HELP0
320 *HELP2
330  PRINT
340  PRINT "オワリ ノ ジコク ヲ ニュウリョク シテクダサイ。"
350 *HELP0
360  PRINT : FLAG=-1
370  ON HELP GOSUB *HELP3 : HELP OFF : HELP ON
380  LINE INPUT WAIT 50,DUMY$
390  RETURN
400 *HELP3
410  PRINT "アル ジコク カラ アルジコク マデ ノ 秒 スウ ヲ モトメマス。"
420  PRINT "ソレゾレ HH:MM:SS ノ カタチデ ジコクヲ シテイシテ クダサイ。"
430  PRINT
440  RETURN 390
```

これをさらにおし進めて，マニュアルは不要，わからない時は HELP キーを押すだけで何でもわかるプログラムを作って見るのも良いでしょう．

また，プログラムでいくつかの項目から選ぶ場合には，次の様なスペースキーとリターンキーしか使わない方法もあります．

まず，次のプログラムを入力して実行させてみましょう．

```
100 '
110 ' one key ノ ニュウリョク
120 '
130 DEF FNX=RND*640 : DEF FNY=RND*200 : DEF FNCOL=RND*6+1
140 SCREEN 0,0 : CLS 2
150 CONSOLE 0,25,0,0 : WIDTH 40,25
160 LOCATE 3,24 : PRINT "box";
170 LOCATE 12,24 : PRINT "circle";
180 LOCATE 23,24 : PRINT "clear";
190 LOCATE 33,24,0 : PRINT "end";
200 K=0
210 '
220 *LOOP
230  COLOR@ (K*10+2,24)-(K*10+7,24),4
240  K$=INPUT$(1)
250  IF K$=" " THEN COLOR@ (K*10+2,24)-(K*10+7,24),0
                :K=(K+1)MOD 4 : GOTO *LOOP
260  IF K$<>CHR$(13) THEN *LOOP
270  COLOR@ (K*10+2,24)-(K*10+7,24),6
280  ON K+1 GOSUB *BOX,*CIRCL,*CRTCLS,*BYE
290  GOTO *LOOP
300 '
310 *BOX
320  LINE (FNX,FNY)-(FNX,FNY),FNCOL,B
330  RETURN
340 '
```

```
350 *CIRCL
360  CIRCLE (FNX,FNY),RND*50+20,FNCOL
370  RETURN
380 '
390 *CRTCLS
400  CLS 2 : RETURN
410 '
420 *BYE : LOCATE ,,1 : END
```

実行させるとまず画面の下の方にメニューが表示されて，最初の項目だけが白黒反転していますね．そこでスペースキーをたたくと反転した項目が移ります．そして適当なところでリターンキーを押すと画面の上部にその項目の名前が表示されます．つまりスペースキーとリターンキーだけで幾くつかの項目から指定する事ができるのです．こんな入力方法も便利だと思いませんか？．

また，せっかく PF キーが10個もあるのですからこれを使って指定する事にしても良いですし，ライトペンを使って入力するなんてシャレてませんか？．

まだまだ他にも考え付きますが，入力方法はキーボードからとは限りませんし，またキーボードだけでもいろいろな方法があります，単に入力と言ってもいくらでも改良の余地はあるのです．

7.5　使い易いプログラムを作るために－出力編－

使い易いプログラムは表示が見易いプログラムでもあります．ここでは画面への出力の仕方について検討しましょう．

画面の出力文としては PRINT 文があります．しかしこの PRINT 文を使うだけで見易い表示をするのは大変です．

例えば数値を出力すると左詰めで表示され，また任意の幅で揃えて出力する事は至難のわざです．

そこで BASIC には書式を指定して出力する文，PRINT USING 文があります．

1. PRINT USING

PRINT USING 文は数値や文字列の表示を指定された書式に従って行う文です．

PRINT USING　〈書式制御文字列〉；〈式〉[；……] [；]

書式の指定は書式制御文字からなる文字列で行います．

① 数値は右詰めに表示する時にはナンバ記号（#）を使います．
次の例を見て下さい．

```
list
100 FORM$="#######"
110 PRINT "  print        |print using"
120 FOR I=1 TO 13
```

```
130   READ X
140   PRINT X,"|";
150   PRINT USING FORM$;X
160 NEXT
170 PRINT "   format is     "+FORM$
180 END
190 DATA 1,12,123,1234,12345,123456,1234567,12345678
200 DATA 123456e-2,123456e-1,123456e0,123456e1,123456e2
Ok
run
  print          |print using
 1               |      1
 12              |     12
 123             |    123
 1234            |   1234
 12345           |  12345
 123456          | 123456
 1.23457E+06     |1234570
 1.23457E+07     |%12345700
 1234.56         |   1235
 12345.6         |  12346
 123456          | 123456
 1.23456E+06     |1234560
 1.23456E+07     |%12345600
  format is       #######
Ok
■
```

数値を PRINT 文と PRINT USING 文とで比較して表示しています．ここでナンバ記号の数は，数値を出力する桁数を指定していますが，数値の桁がこれを越える場合には数値の直前にパーセント記号（%）が表示されます．

また，この場合値が実数値の場合には小数点以下を四捨五入しています．

② 数値の固定小数点表示には「#」と「.」を組み合わせて使います．

```
run
  print          |print using
 1               |     1.0000
 12              |    12.0000
 123             |   123.0000
 1234            |  1234.0000
 12345           |12345.0000
 123456          |%123456.0000
 12345.7         |12345.7000
 1234.57         | 1234.5700
 123.456         |  123.4560
 12.3456         |   12.3456
 1.23456         |    1.2346
 .123456         |    0.1235
 .0123456        |    0.0123
  format is       #####.####
Ok
■
```

小数点の位置はピリオドによって示し，それより右が小数部分，左が整数部分の桁指定となります．

また，この場合も整数部分が桁数を越えた時には「%」が表示されます．しかし，小数部分の桁数のみが多い場合には「%」は表示されず，単に指定された桁以下を丸めます．

③ 3桁ごとにコンマ「,」を付けて数値を表示する場合には「,」を使います．

```
run
  print          |print using
 1               |          1
 12              |         12
 123             |        123
 1234            |      1,234
 12345           |     12,345
 123456          |    123,456
 1.23457E+06     |  1,234,570
 1.23457E+07     | 12,345,700
 1234.56         |      1,235
 12345.6         |     12,346
 123456          |    123,456
 1.23456E+06     |  1,234,560
 1.23456E+07     | 12,345,600
  format is       #########,
Ok
■
```

そして固定小数点表示においても同じ事ができます．ただしコンマで区切られるのは整数部分です．

また，この時コンマは，ピリオド（小数点）よりも左側に書かなければなりません．

```
run
  print          |print using
 1               |          1.000
 12              |         12.000
 123             |        123.000
 1234            |      1,234.000
 1234.5          |      1,234.500
 1234.56         |      1,234.560
 1234.57         |      1,234.570
 1234.57         |      1,234.570
 123.456         |        123.456
 1234.56         |      1,234.560
 12345.6         |     12,345.600
 123456          |    123,456.000
 1.23456E+06     |1,234,560.000
  format is       ########,.###
Ok
■
```

④ 円記号「¥」を数値の先頭に表示するには「¥¥」を用います．これは金額の表示に便利です．

```
run
  print          |print using
 1               |          ¥1
 12              |         ¥12
 123             |        ¥123
 1234            |      ¥1,234
 12345           |     ¥12,345
 123456          |    ¥123,456
 1.23457E+06     |¥1,234,570
 1.23457E+07     |%¥12,345,700
 1234.56         |      ¥1,235
 12345.6         |     ¥12,346
 123456          |    ¥123,456
 1.23456E+06     |¥1,234,560
 1.23456E+07     |%¥12,345,600
  format is       ¥¥########,
Ok
■
```

ただし「¥」と円記号が１文字の場合は書式制御文字とはなりません．

⑤ 数値の前の余白の全てにアスタリスク「＊」を表示させるには，「＊＊」を用います．これは預金通帳の金額表示でお馴染みですね．

```
run
  print         |print using
 1              |********1
 12             |*******12
 123            |******123
 1234           |****1,234
 12345          |***12,345
 123456         |**123,456
 1.23457E+06    |1,234,570
 1.23457E+07    |%12,345,700
 1234.56        |****1,235
 12345.6        |***12,346
 123456         |**123,456
 1.23456E+06    |1,234,560
 1.23456E+07    |%12,345,600
  format is      **######,
Ok
■
```

この場合にも円記号と同様「＊」１文字ではだめです．

⑥ 数値の前に「¥」を表示し，その前の余白は全て「＊」とする（④と⑤の組み合わせ）の場合には「＊＊¥」を使います．

```
run
  print         |print using
 1              |********¥1
 12             |*******¥12
 123            |******¥123
 1234           |****¥1,234
 12345          |***¥12,345
 123456         |**¥123,456
 1.23457E+06    |¥1,234,570
 1.23457E+07    |%¥12,345,700
 1234.56        |****¥1,235
 12345.6        |***¥12,346
 123456         |**¥123,456
 1.23456E+06    |¥1,234,560
 1.23456E+07    |%¥12,345,600
  format is      **¥######,
Ok
■
```

⑦ 指数表示を行う場合には桁数指定の「#」並びの後に「^^^^」を付けて使います．

```
list
100 *LOOP
110   INPUT "number ";NUM!
120   LINE INPUT "format ? ";FORM$
130   PRINT "print","|print using"
140   PRINT NUM!,"|";: PRINT USING FORM$;NUM!
150   PRINT "format is"," ";FORM$
160   PRINT
170   GOTO *LOOP
Ok
run
number ? 1.2345e23
format ? ##.####^^^^
print         |print using
 1.2345E+23   | 1.2345E+23
format is      ##.####^^^^
```

```
number ? -9.8765e-21
format ? ##.####^^^^
print           |print using
-9.8765E-21     |-9.8765E-21
format is        ##.####^^^^

number ? ■
```

⑧　数値の符号の表示を指定するには「＋」又は「－」を使います．

「＋」の場合は書式制御文字列の最初，または最後に付けた時にその位置に数値の符号が表示されます．

「－」の場合は書式制御文字列の最後に付ける事で，数値が負の時だけ数値の後に「－」が表示されます．

```
run
number ? 123456
format ? +#######
print           |print using
 123456         | +123456
format is        +#######

number ? -987654
format ? +########
print           |print using
-987654         |  -987654
format is        +########

number ? ■
```

ただし，この「＋」や「－」を２つ以上並べた場合には，制御文字以外としてそのまま表示されます．

⑨　アンダーバー「＿」を使うと次の１文字は常に表示されます．つまりアンダーバーは書式制御文字列におけるエスケープ·キャラクタで，これを用いて「#」等の制御文字そのものを出力する事が出来ます．

```
run
number ? 3.1416
format ? pai=#.#####_#_#
print           |print using
 3.1416         |pai=3.14160##
format is        pai=#.#####_#_#

number ? 50
format ? ﾃｲｶ ¥¥###_+-ｲｸﾗｶﾃﾞ ｳﾙｮ
print           |print using
 50             |ﾃｲｶ   ¥50+-ｲｸﾗｶﾃﾞ ｳﾙｮ
format is        ﾃｲｶ ¥¥###_+-ｲｸﾗｶﾃﾞ ｳﾙｮ

number ? ■
```

⑩　「&」を使ってn個（n≧0）の空白を囲んだ場合には「&」から「&」までに＋２文字分の領域に与えられた文字列を左づめで表示します．また，この特殊な場合として「！」は与えられた文字列の頭１文字だけ表示します．

```
list
100 *LOOP
110   LINE INPUT "string ? ";S$
120   LINE INPUT "format ? ";FORM$
130   PRINT "print","|print using"
140   PRINT S$,"|";: PRINT USING FORM$;S$
150   PRINT "format is"," ";FORM$
160   PRINT
170   GOTO *LOOP
OK
run
string ? ｸﾓﾘ ﾉﾁ ﾊﾚ
format ? ｱｽ ﾉ ﾃﾝｷ ﾊ & & ﾃﾞｼｮｳ｡
print          |print using
ｸﾓﾘ ﾉﾁ ﾊﾚ      |ｱｽ ﾉ ﾃﾝｷ ﾊ ｸﾓﾘ ﾃﾞｼｮｳ｡
format is       ｱｽ ﾉ ﾃﾝｷ ﾊ & & ﾃﾞｼｮｳ｡

string ? ﾍﾞｰｼｯｸ
format ? ﾜﾀｼ ﾊ ﾍﾞｰｼｯｸ ﾉ ! ﾉ ｼﾞ ﾓ ﾜｶﾘﾏｾﾝ｡
print          |print using
ﾍﾞｰｼｯｸ         |ﾜﾀｼ ﾊ ﾍﾞｰｼｯｸ ﾉ ﾍ ﾉ ｼﾞ ﾓ ﾜｶﾘﾏｾﾝ｡
format is       ﾜﾀｼ ﾊ ﾍﾞｰｼｯｸ ﾉ ! ﾉ ｼﾞ ﾓ ﾜｶﾘﾏｾﾝ｡

string ? ■
```

⑪ 文字列を代入する場合は「@」を使います.

ただし，1つの「@」に対して1つの文字列全体を与え，また「@」の個数が与えた文字列の個数より多い場合には残りの「@」は無視されます.

```
run
string ? ﾆｬﾝｺ
format ? ﾜﾀｼ ﾊ @ ﾃﾞｽ
print          |print using
ﾆｬﾝｺ           |ﾜﾀｼ ﾊ ﾆｬﾝｺ ﾃﾞｽ
format is       ﾜﾀｼ ﾊ @ ﾃﾞｽ

string ? ■
```

以上，PRINT USINGの使い方について説明しましたが，この他にも，TAB, SPCを使ってもかなり美しい出力ができますし，セパレータのコンマを使っただけでも効果があります. 要は，プログラマのセンスというところでしょうか？

8章 うまいデバッギングのために

プログラムを作っていると必ずといっていいほどバグ（エラー虫）が住みつきます．そこでこれを取り除く作業，デバッグ（エラー虫を取り除く）をする必要がある訳ですが，この虫取り作業はプログラムの開発期間のかなりの部分を費やします．そこでプログラムを作る時に，バグが発生しないように注意する，と考えてもそうはいきません．計算機を動かすプログラムは，しょせんは人間の作るもので誤りをなくす事は出来ないのです．

さて，このバグにもいろいろあり，タイプミスや感違いにより起こる文法ミスから，アルゴリズムに誤りがある論理ミスまでさまざまです．文法ミスの場合は救いがあり，エラーのあった行を見ればすぐに解ります．しかし，論理ミスの場合はそうはいかないのです．無限ループになっていたり，エラーメッセージが出ないけれど結果が無茶苦茶であったり，最悪の場合には，一見まともな結果を出すが，ある特定の条件でおかしくなったりもします．そこで，ここではプログラムからバグを取り除く方法について述べてみます．

8.1 デバッギングの基本

1. エラー発生行の表示

まず，エラーの発生した行を探さなければなりません．エラーが発生すると，BASICはどんなエラーが発生したかのエラーメッセージと，それが発生した行番号を表示します．Syntax Error（文法ミス）の時は，さらにその行が表示されますし，それ以外の時でもHELPキーを押す事でエラーの発生した行を表示します．

またこの他にも，

```
LIST .
EDIT .
```

などでその行を表示したり，エディットする事が出来ます．

2. 実行をトレースする (TRON／TROFF)

N₈₈-BASICには，プログラムの実行の流れを表示するトレースモードがあります．このトレースモードの時は，プログラムの一行を実行するごとに，その行番号を表示します．

トレースモードにするには，

TRON

解除するには，

TROFF

をそれぞれ実行します．

```
new
Ok
100 for i=0 to 30 : gosub *sub : next
110 end
120 *sub
130  beep : print "nyaou~ ";
140  return
run
nyaou~ nyaou~ nyaou~ nyaou~ ^C
Break in 130
Ok
tron
Ok
cont
nyaou~ [135][140][120][130]nyaou~ [135][140][120][130]nyaou~ [135][140]^C
Break in 140
Ok
troff
Ok
■
```

トレースモードの状態では，プログラムの実行と行番号の表示を同時に行いますから，多少画面の表示が見にくくなるかもしれません．また行番号の表示のために実行速度が少し落ちます．もし途中で止める場合には一般にプログラムを停止させる時と同様, CTRL-S を押して下さい．また CTRL-O で通常の画面への出力が禁止されますが，トレースモードでこのキーを押した場合には行番号の表示も同時に禁止されます．

なお，トレースモードは，プログラムを書き替えても解除されません．TROFF か NEW 以外にトレースモードを解除する方法はありません（NEW を行った場合は，もちろんプログラムは消されてしまいます）．

8.2 デバッギングのノウハウ

さて，現実の虫取り作業はどの様に行うか，そのヒントを述べましょう．

1. サブルーチンごとにチェックする

いきなり全部のプログラムをデバッグする事はできません．サブルーチンの1つ1つを，それ自体ではバグなしに動作する様にしていき，しだいにメインルーチンへ近ずく様にして虫を追い込みましょう．

2. TRON／TROFF を利用する

プログラムの流れをつかむためには，トレースモードの利用が一番です．まずプログラムの流れが正しいかどうかチェックしましょう．

3. 変数の値をチェックする

Illegal function call(?FC Error) 等のエラーが発生した時は，その引数が間違っています．引数が変数の時は，その値を PRINT 文で表示させてみましょう．

4. 配列は正しく使っているか？

よくある間違いとして起こるのが，配列の添字の値の誤りから起こるエラーです．また DIM 文を使って配列は正しく宣言されていますか？

5. STOP 文や PRINT 文を使う

プログラム中に STOP 文を含め，プログラムの実行を少しずつ行いながら，PRINT 文でその時の変数の値や IF 文での流れをチェックしましょう．

6. REM 文を利用する

エラーの原因となりそうな行があったら，その行を REM 文でコメントに変えて，プログラムの実行がどの様に変化するか調べてみましょう．

7. ループを繰り返す回数をカウントする

プログラム中でループは正しく回っていますか？何回ループを回っているか，新しい変数を使ってカウントしましょう．

8. モデルデータを入力する

入力する値でプログラムの実行がどの様に変化するかを調べてみると，エラーの原因がつかめる事があります．正しい結果の予想のつくデータを入力して，実際の結果と比較しましょう．

9. コーヒーでも飲んで気分を変える

これは最後の手段ですが，根をつめてデバックをしても効果はありません．感違いをしていて

バグが見つからない事もあるからです．一服して気分を落ち着けましょう．

また2・3日してからプログラムを読み直すとバグを発見できる事もよくあります．

8.3 エラー，その傾向と対策

これまでで，論理エラーを含む一般的なエラーについてのデバッグ法を話してきましたが，ここではエラーメッセージが表示された場合について，そのエラーの原因と対策を考えてみます．

N_{88}-BASICのリファレンス・マニュアルが手もとにある方は，ご覧になれば分かると思いますが，N_{88}-BASICには，実に多くのエラーメッセージが用意されています．ここではその全てについては説明できませんが，この中で起こる可能性が高いものをピックアップしました．これだけのことを覚えておけば，まずほとんどのエラーに対しては対処できるでしょう．

ただし，ここで解説するエラーメッセージは，ディスク入出力関係も含んでいますから，N_{88}-DISK BASICのメッセージをメインにしています．ROM BASICの方は，フルメッセージの後のカッコ内の省略型のメッセージで参照して下さい．

エラーコード：1　NEXT without FOR (?NF Error)

意味：FOR文と対応していないNEXT文がでてきた．

原因：①FOR文の数とNEXT文の数が合っていません．

→両者の数や制御変数とが合っていますか．

例
```
FOR I=…
     ⋮
  NEXT J ←FOR 文と対応していない NEXT 文
  NEXT 1
```

この例ではNEXT Jでエラーが起こります．

②他からFOR～NEXTループの中に飛び込んでいます．

→ループの外からFOR～NEXTループの中に分岐しているGOTO文やGOSUB文がないかどうか調べてみてください．

例
```
100 GOTO 210
       ⋮
200 FOR I=…
210 ・・・
220 NEXT I
```

これは，200～220行のFOR～NEXTループの中にGOTO文で飛び込んでいます．こ

れは100行の GOTO 文の飛び先が間違っているとしか考えられません．GOTO 文の前のプログラムから考えて，GOTO 文の飛び先を直してください．

ワンポイントアドバイス：

エラーの起きた行の NEXT 文について考えます．必要か不必要か，制御変数が合っているか，対応する FOR 文はどこにあるかなど考えて，そして，抜けている FOR 文を追加したり，余分な NEXT 文を消除して，FOR～NEXT を正しく対応させて，1つのプログラム中の FOR 文の数と NEXT 文の数を合うようにします．

エラーコード：2　Syntax error (?SN Error)

意味：プログラムの構文（文の書き方）が間違っています．

原因：①プログラム中に規定の予約語（キーワード）以外のものがあります．

→大半はタイプのミスですから，正しいコマンド，ステートメントに修正します．

例　PLINT　→PRINT が正しい．
　　IMPUT　→INPUT が正しい．
　　ATAN　→ATN が正しい．

②関数が代入文の左辺にあったり，ステートメントのように単独で使われています．

→関数は代入文の右辺または PRINT 文中または式の中で使用します．

例　ABS(10)=I+J
　　CHR$(5)=A*B

これらは誤りです．ABS(10) や CHR$(5) などは，変数に使えません．

③変数名が英字で始まっていない．または，許されない記号を含んでいます．

例　@PR, FNT などは許されない変数名です．変数名の規則をよく読んでみてください．

④複文（マルチステートメント）の区切り記号「：」が抜けています．

→「；」と間違えていませんか．「：」を忘れていませんか．エディット中に誤って消してしまうことがあります．

例　PRINT PRINT I
　　→これは PRINT とPRINT I の間に「：」が抜けています．

⑤TAB, SPC 関数を PRINT 文中以外で使用している．

例　K$=SPC(10)+〝*〞

などは誤まり．

⑥ERR, ERL, CSRLIN などの予約変数に値を代入しようとした．

例　ERR=10
　　CSRLIN=A*B

などは誤まり.
⑦行番号が許される範囲（1～65529までの整数）にありません.

例　100000 PRINT I

は行番号が大き過ぎます.
⑧IF 文中で対応する THEN のない ELSE が使われています.
→IF の数と THEN の数を合うようにします.

例　IF I<J ELSE …

は THEN がないので誤り.
⑨関数の引数の数が一致していません.
→各関数の引数の数を確かめましょう.

例　PRINT INSTR("ABC")

は誤り. INSTR 関数を見てください. 引数の数が合っていません.
⑩ステートメントの引数の数が一致していません.
→各ステートメントの引数の数を確めましょう.

例　PRESET(100, 10, 5)

は誤り, PRESET を見てください. 引数の数が合っていません.
⑪ラベルの使い方が間違っている.
　ラベルのチェックは, プログラムの実行に先立って行われますから, ラベルに文法エラーがあった場合は, 行番号を伴わない Syntax エラーが表示されます. この場合は, プログラムの最初からラベル名をチェックして下さい. おそらく許されないラベルが使われています.

例　GOTO *123

ラベル名の最初は英字でなければならない.

ワンポイントアドバイス：
このエラーが起きた時は, LIST . や EDIT . などで, そのエラーの起きた行を表示させてよく調べてみてください. 多くの場合, 原因はその行にあります. 特に原因①の場合が多いようです.

エラーコード：3　RETURN without GOSUB (?RG Error)

意味：サブルーチンとして分岐していないのに，RETURN 文に出合った．

原因：①GOTO 文でサブルーチンへ分岐している．

→GOSUB 文に直す．

②メインプログラムとサブルーチンが分離していなく，メインプログラムの実行を終えた後，自動的にサブルーチンを実行して，RETURN 文に出合った．

→メインプログラムの終りには END 文を，サブルーチンの終わりには RETURN 文を必ず置きましょう．

ワンポイントアドバイス：

サブルーチンのネストの深さに注意し，また，メインプログラムの終りには END 文を必ず置くようにしましょう．

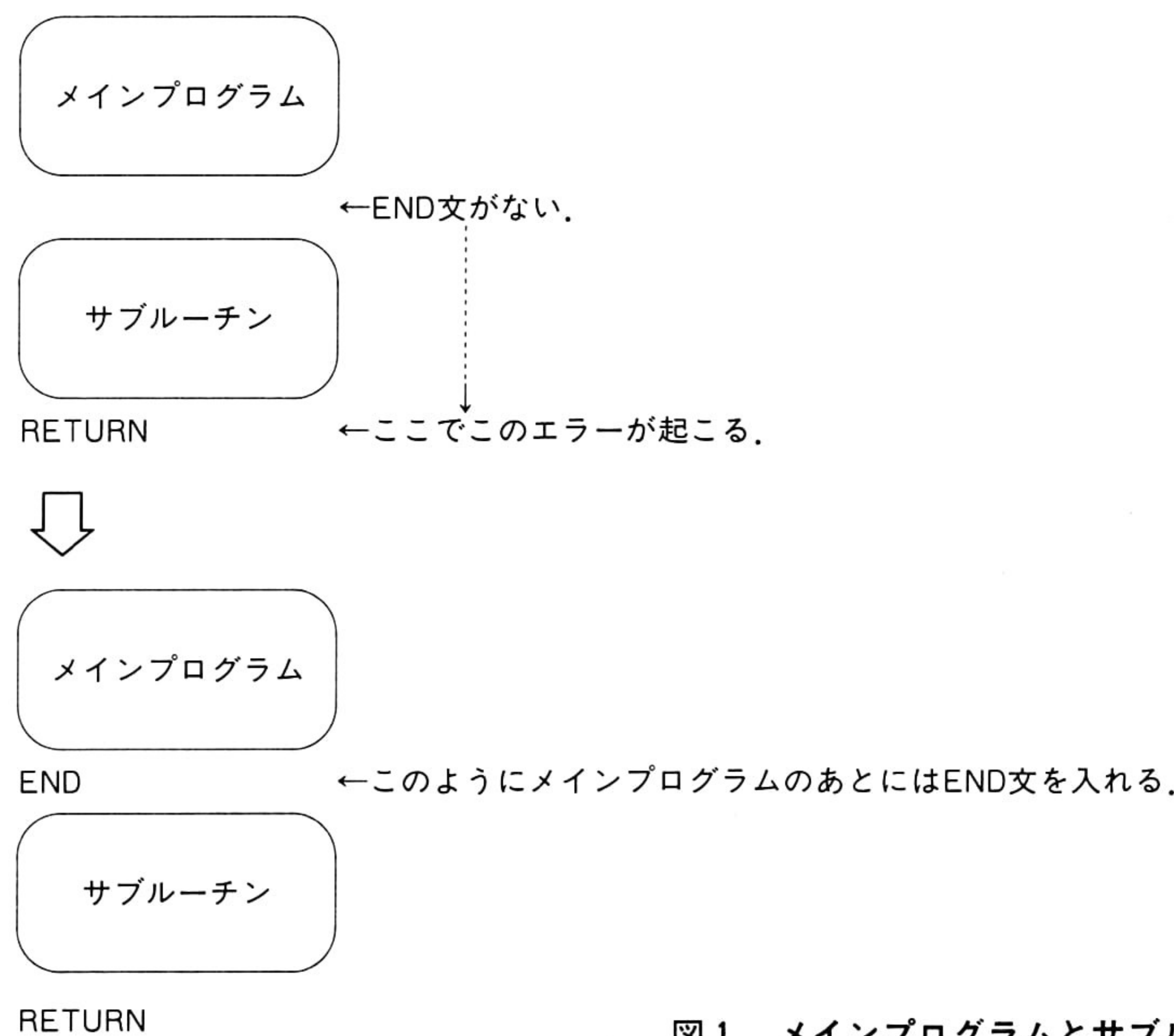



図1　メインプログラムとサブルーチン

エラーコード：4　Out of data (?OD Error)

意味：READ 文によって読まれるべき DATA 文中のデータが不足した．

原因：①RESTORE 文の使い方が合っていません．

→間違った場所を指していないか調べましょう．

例
```
10 RESTORE 1100
        :
110 DATA 100, 200, 300
```

1100 READ Y

RESTORE で指定された行番号以降に DATA 文がないので，10行実行後，READ 文を実行するとエラーとなります．この場合，10行のRESTORE 文の1100が間違っています．

②データの数が十分にありません．

→READ 文で読むデータの個数と DATA 文で用意するデータの個数を同じにしましょう．

ワンポイントアドバイス：

まず，データの個数や順番が間違っていないかをみましょう．さらに，RESTORE 文の使いかたを確かめましょう．DATA 文がないこともありますよ．

エラーコード：5　Illegal function call (?FC Error)

意味：関数やステートメントの呼び方が間違っている．

原因：①指定したパラメーターの値が，許される範囲にない．

例　COLOR 8

②配列の添字の値が負になっている．

③定義されていない USR 関数を使用した．

④SWAP において，定義されていない配列や変数を使用した．

⑤PRINT USING において桁指定が，24桁を越えている．

⑥ASC関数において，ヌルストリングを指定した．

ワンポイントアドバイス：

このエラーは，Syntax error と並んで最もよくお世話になるエラーでしょう．症状はかなり広範囲にわたりますので，各ステートメント，各関数の正しい使い方をもう1度見直すしか手がありません．早く正しいパラメータの与え方を覚えるようにして下さい．

エラーコード：6　Overflow (?OV Error)

意味：入力された数値，代入される数値，演算結果などが許される範囲を越えています．

原因：①整数演算の結果，あるいは引き数が整数でなければならないのに—32768〜32767の範囲にない．

→必要であれば実数型変数を使用します．また，—32768〜32767の範囲に収まるようにします．

例　RANDOMIZE 35436

A=54321: B%=CINT(A)

共に32767より大きいので誤り．

②実数演算などの結果が約1. 7E38～約−1. 7E38の範囲にありません.

例 A=20E20*20E20

は Overflow エラーが起こります.

ワンポイントアドバイス：

これもエラーコード5 Illegal function call と同様に，エラーの起きた行の変数値や定数値を調べてみましょう．おかしな値になっていませんか．おかしな値だったら，その原因を考えてみましょう．

エラーコード：7　Out of memory (?OM Error)

意味：メモリ容量が足りません.

原因：①プログラムが長すぎて規定のメモリ領域に収まらない.

→プログラムリスト中の余分な空白を除き，注釈文をできるだけ除きましょう．このとき，変数の区別などの，必要な空白は取り除いてはいけません.

例
```
100  FOR I=0 TO 100  STEP 1
110  MEM(I)=I
120  NEXT I

→100  FOR I=0 TO 100: MEM(I)=I :NEXT
```

このようにすると，19バイトほどプログラムが小さくなって，その分メモリを有効に使えるようになります.

②プログラムは収まっても，変数などの領域をとると，メモリ容量が不足する.

→次々と新しい変数を使用しないで，前に使用した後不要となった変数を使用するようにしましょう．また①に習って，プログラムを小さくすることも考えましょう.

例
```
100 FOR I=1 TO 100
        ⋮
150 NEXT I
160 FOR J=1 TO 100
        ⋮
210 NEXT J
```

このプログラムで，160～210行の FOR～NEXT ループでも，100～150行の FOR～NEXT ループと同じ制御変数 I を使うようにすると，6バイトほど小さくなります.

③配列が大き過ぎて，メモリ上にその領域をとることができなくなった.

→必要以上に大きく配列をとっていませんか．例えば要素数が，5つあればいいのに，DIM文を使用しないで，メモリの無駄が生じていませんか．

例
```
 10 DIM A(4)
       ⋮
100 A(1)=100
```

10行を省略すると13バイト程無駄となります．

④BASIC 内部のスタックが一杯になってしまった．

→サブルーチンや FOR～NEXT ループなどのネスティングが深すぎるかを調べてみましょう．

例
```
100 …
110 GOSUB 100
     ⋮
200 RETURN
```

このプログラムを実行すると，Out of memory エラーが起こります．これは明らかに110行の GOSUB 文の飛び先が間違っています．

注：スタックには，サブルーチン実行後のもどり先や FOR～NEXT ループの開始位置などが積まれます．

ワンポイントアドバイス：

いずれの場合も FRE関数を使ってメモリの残り量を確認してみましょう．余裕がかなりあるのに Out of memory エラーを起こすようであればスタック（④）が原因です．そうでなければプログラムを短くしたり，配列を整数型にできないかを考えてみましょう．

エラーコード：8　Undefined line number (?UL Error)

意味：行番号が指定されていない．指定された行番号が存在しない．

原因：①GOTO, GOSUB などの分岐先の行番号，またはラベルが存在しない．

→エラーの起きた行を表示させてみて，GOTO 文，GOSUB 文の飛び先を知りましょう．そして，その飛び先があるか確認しましょう．

②RESTORE, RUN, RETURN で指定した行番号または，ラベルが存在しない．

③RENUM 実行時に参照される行番号が存在しない．

→①や②と同じ，GOTO 文などの飛び先の行番号またはラベルや，RESTORE 文で指定した行番号またはラベルがない．

エラーコード：9　Subscript out of range (?BS Error)

意味：配列の添字が，0からDIM文によって宣言した大きさの範囲内にない．

原因：①演算の結果，配列の添字が大きくなり過ぎていた．

→添字の値をPRINTして確認してみましょう．

②配列の次元数を誤っています（添字の個数を間違っています）．

例　I=A(B, C, D)

3つでいいのですか．

DIMで宣言したときの添字の個数と比べてみてください．同じでなければいけません．

ワンポイントアドバイス：

FOR～NEXT文で添字の値を変化させる時に注意してください．添字が負になったら，Illegal function callエラーを，大き過ぎたら，Subscript out of rangeエラーを起こします．

エラーコード：10　Dupulicate Definition (?DD Error)

意味：同じ名前の配列やFN関数を2度宣言しています．

原因：①同じDIM文やDEF FN文を複数回実行しようとしています．

→FOR～NEXTループの中や，GOTOループの中にDIM文やDEF FN文があってはいけません．

例
```
10 FOR I=1 TO 10
20 DIM A(8)
30 NEXT I
```

これではいけません．このようなプログラムは意図して作成するはずがありませんから，エディット中に行番号の付け間違いをしてしまったようです．

②BREAK後GOTO文でプログラムを再開してDIM文やDEF FN文に出会った．

→STOPキーでプログラムの実行を中断しても変数や配列宣言はそのまま保存されていますので，プログラムの実行を再開するときは充分注意してください．

ワンポイントアドバイス：

どんな配列や関数を使用しているかを明確にするためにも，DIM文やDEF文は全てプログラムの先頭にいれるべきです．また，必要ならばCLEAR文を行って既存の配列や変数などを消去すること．

エラーコード：11　Division by Zero (?/0 Error)

意味：0で割り算を行おうとした．

原因：①除算の除数が0になった．または，定義していない変数で除算を行ってしまった．

→エラーの起きた行の除数を確かめてみよう．

②関数 TAN での引数がπ／2になっていた．

ワンポイントアドバイス：

必要ならば演算前に値のチェックを行ってエラーを回避するようにしよう．

エラーコード：12 Illegal direct (?ID Error)

意味：ダイレクトモードでは使えないステートメントを使用した．

原因：①ダイレクトモードで，DEF FN 文により関数を定義しようとした．

エラーコード：13 Type mismatch (?TM Error)

意味：変数の型がくい違っています．

原因：①文字型変数に数値を代入しようとした．

②数値型変数に文字列を代入しようとした．

③(FN も含む) 関数の引き数の型が合っていない．

→CHR$ と ASC, STR$ と VAL の混同に注意してください．また，文字型でない FN 関数を文字型で扱っていませんか．

④FOR 文での制御変数に倍精度実数型を使ってしまった．

→制御変数を単精度実数型または整数型に直します．

エラーコード：15 String too long (?LS Error)

意味：文字列が長すぎる．

原因：①文字型変数に256文字以上の文字列を代入しようとした．②演算の結果，文字列が256文字以上となってしまった．

ワンポイントアドバイス：

N_{88}-BASIC で扱うことができるストリングの最大長は，255文字です．それ以上の長さの文字列を扱う必要がある時は，2つあるいはそれ以上の部分に分割して処理することを考えなければいけません．

エラーコード：17 Can't continue (?CN Error)

意味：CONT 文によるプログラムの続行ができない．

原因：①BASIC の内部ポインタを破壊した．

→プログラムの文面に変更を加えるとポインタが破壊されてしまいます．

②INPUT 文，PAINT 文などの実行中に STOPした．

エラーコード：18　Undefined user function (?UF Error)

意味：定義されていない関数が参照された．

原因：①DEF FN 文で定義されていないユーザー定義関数を参照しようとした．

②配列の名前が予約語 FN で始まっていて，ユーザー関数と解釈された．

ワンポイントアドバイス：

FN 関数を使用する時は，それ以前に DEF FN 文によってその関数を定義しておかなければなりません．ですから，使用する FN 関数はプログラムの文頭で DEF FN 文によって定義しておきましょう．

エラーコード：19　No RESUME (?NR Error)

意味：RESUME 文がない．

原因：①エラー処理ルーチンが RESUME 文または END 文で終わっていない．

ワンポイントアドバイス：

エラー処理ルーチンは RESUME, END, ON ERROR GOTO 0 のいずれかで終っていなければなりません．

エラーコード：20　RESUME without error (?RW Error)

意味：エラーがないのに RESUME 文に出合った．

原因：①GOTO 文やGOSUB 文でエラー処理ルーチンの中へ分岐している．

→GOTO 文や GOSUB 文の飛び先が間違っています．

②メインプログラムとエラー処理ルーチンが分離していなくて，メインプログラムの実行終了後，自動的にエラー処理してRESUME 文に出会った．

→メインプログラムとエラー処理ルーチンの間に END 文を入れて区別しましょう．

エラーコード：22　Missing operand (?MO Error)

意味：必要なオペランドが欠けている．

原因：①ステートメントや関数に必要とされるパラメータが指定されていない．

→各ステートメントや関数の使い方を確認し直してください．

②代入文の右辺がない．

例
```
LET A=
```

③演算子があるのに被演算子が欠けている．

例
```
A=B*
C=D^
```

エラーコード：26　FOR without NEXT (?FN Error)

意味：FOR 文と対応する NEXT 文がない．

原因：①FOR 文の数と NEXT 文の数が合っていない．

→両者の数や制御変数とが合っていますか．

```
例 100 FOR I=… ←対応するNEXT 文がない．
          ⋮
   140 FOR I=…
   150 NEXT J ←制御変数がくい違っている．
```

ワンポイントアドバイス：

エラーの起きた行の FOR 文について考えます．対応する NEXT 文はどこにあるか，制御変数が合っているかなどを考えて，そして，抜けている NEXT 文を追加したり，余分な FOR 文を削除したりして，FOR～NEXT を正しく対応させて，1つのプログラム中の FOR 文の数と NEXT 文の数を合うようにしましょう．

エラーコード：27　Tape read error (?TP Error)

意味：テープからのデータ読み込み時にエラーを生じた．

原因：①カセットテレコの音量が適当でない．

②カセットテレコとの相性が良くない．

ワンポイントアドバイス：

音量を細かに調整する．一般的に，音量が小さいとファイルを見つけることができず，大きいとこのエラーが起こるようです．

②もし〝CAS1:〟を使っているのであれば，〝CAS2:〟にして転送レートを落して，SAVE からやり直します．最後の手段としては，他のテレコと交換します．

エラーコード：29　WHILE without WEND (DISK BASIC専用エラー)

意味：WHILE 文に対応する WEND 文がない．

原因：①WHILE 文の数と WEND 文の数とが合っていない．WHILE が多い．

→両者の数が合っていますか．

```
例 100  WHILE … ←対応する WEND 文がない．
            ⋮
```

ワンポイントアドバイス：

エラーの起きた行の WHLLE 文と対応する WEND 文はどこにありますか．抜けている WEND 文を追加し，WHILE～WEND を正しく対応させて，1つのプログラム中の

WHILE 文の数と WEND 文の数を合うようにします.

エラーコード：30　WEND without WHILE (DISK BASIC専用エラー)

意味：WEND 文に対応する WHILE 文がない. WEND が多い.

原因：①WHILE 文の数と WEND 文の数とが合っていない.

→両者の数が合っていますか.

例
```
           :
  150 WEND ←WHILE 文と対応していないWEND 文.
```

この例では150行の WEND 文で WEND without WHILE エラーが起こります.

②他から WHILE～WEND ループの中に飛び込んできている.

→ループの外から WHILE～WEND ループの中に分岐している GOTO 文や GOSUB 文がないかどうかを調べてください.

例
```
100 GOTO 210
        :
200 WHILE …
210     :
220 WEND
```

これは200～220行の WHILE～WEND ループの中に GOTO 文で飛び込んできています. それは, 100行の GOTO 文の飛び先が間違っているとしか考えられません. GOTO 文の前のプログラムから変えて GOTO 文の飛び先を直してください.

ワンポイントアドバイス：

エラーの起きた行の WEND 文と対応する WHILE 文はどこにありますか. 余分な WEND 文を削除して, WHILE～WEND を正しく対応させて, 1つのプログラム中の WHILE 文の数と WEND 文の数を合うようにしましょう.

エラーコード：31　Dupulicate label (?DU Error)

意味：同じラベル名が2度宣言されている.

原因：参照される方のラベル名に全く同じものがある.

例
```
100 GOTO *EXIT
        :
200 *EXIT:PRINT…
        :
300 *EXIT:END
```

この例では, 200行と300行に同じラベル名がある.

エラーコード：32　Undefined label (?UN Error)

意味：定義されていないラベル名によって参照しようとした．

原因：①参照される方のラベル名を忘れている．

②同じにしたつもりが，タイプミスなどで合致していない．

エラーコード：52　Bad file number (?BN Error)

意味：不適当なファイル番号を使用した．

原因：①起動時の〝How many files ?〟で答えた数より大きなファイル番号を使用した．

②オープンしていないファイル番号を使用した．

エラーコード：53　File not found (?FF Error)

意味：指定されたファイル名が見つからない．

原因：①ファイルディスクリプタを間違えた．

→デバイス名はあっているか．ファイル名は合っているか．デバイス名とファイル名が間違っていないか．

例　LOAD〝CAS1:TEST〟としようとしたのに，

LOAD〝CAS0:TEST〟としてしまった．

エラーコード：54　File already open (?AO Error)

意味：ファイルを２重にオープンしようとした．

原因：①ループの中に OPEN 文がありませんか．

②同じファイル番号で２度オープンしていませんか．

エラーコード：55　Input past end (?EF Error)

意味：ファイルの全てのデータを読んだ後に，Input 文を実行していた．

原因：データ終了のチェックをしていない．

ワンポイントアドバイス：

シーケンシャルファイルの場合は EOF 関数を，ランダンファイルの場合は LOF 関数を使ってファイルの終りを調べる．

例
```
100 IF EOF(1) THEN 200
110 INPUT #1, A
120 GOTO 10
200 CLOSE #1
```

エラーコード：60　File not Open (?CF Error)

意味：オープンしていないファイル番号を使用した．

原因：OPEN 文でオープンしていないファイル番号を使用しています．

エラーコード：64　Disk I/O Error (DISK BASIC専用エラー)

意味：使用中の入出力装置において，リード／ライトエラーが生じた．

原因：ディスクのアクセス中にフタを開けたなど．

エラーコード：69　Bad allocation table (DISK BASIC専用エラー)

意味：ディスケットの FAT(File Allocation Table) が壊されている．

原因：①DSKO$ 命令の操作を間違って，意味のないデータをディスクに書き込んだ．

②ディスケットに外傷がついた．

→このエラーは致命的で，対処のしようがありません．常に大切なディスケットは，バックアップを作っておきましょう．

8.4 ラベル・クロスリファレンス

さて，最後になりましたが，ツールとして1つだけプログラムをあなたにお贈りしましょう．

N88-BASICでは，行番号のかわりに，ラベルを使ってその行番号の参照が簡単にできます．今までに，アセンブリ言語を使った経験のある方ならお分かりのように，ラベルを無造作にたくさん使った場合，実際にそのラベルのある場所とか，そのラベルを参照している場所を知りたいことがあります．そのような時には，是非このプログラムを利用してください．ただし，このプログラムを使う上での注意点が幾つかありますので，気を付けてください．

このプログラムは，N88-DISK BASIC用に作られています．残念ながら，ディスク装置をお持ちでない方は利用できませんのであしからず．

ディスク装置をお持ちならば，とにかくプログラムを走らせられる状態にします．あとは，実行例を見ながら説明していきましょう．とにかく，プログラムをrunさせると，ファイル名の入力を要求してきますので，答えてください．その時に，必ずそのファイル名で指定したファイルは，アスキーセーブされていなければなりません．つまり，

```
SAVE "prog.asc", a
```

の書式でセーブしておきます．では，アスキーセーブ以外のファイルを指定した場合にはどうなるでしょう．多分，何らかのエラーを起こしてプログラムは停止すると思いますが，このエラーに関するサポートは一切していませんので，気を付けてください．

また，ファイル名としては，ドライブ番号の指定もできます．存在しないファイル名を指定し

た場合には，その旨を表示して再度入力を要求してきます．ディスク装置のフタが開いていたりして，ディスク装置がオフラインの状態にある場合にも，同じようにその旨を表示して，

Retry?

と聞いてきますので，この状態の時にRETURNキーを入力すれば，リトライしてくれますので，その前に，フタが開いているのであれば，閉めます．また，ディスケットがセットされていないのであれば，セットします．しかし，この状態には制限時間があります．Retry ? が表示されてから，約5秒以内にRETURNキーが押されなければ，プログラムは終了してしまいますので，気をつけてください．

さて，無事にファイル名の指定ができると，クロスリファレンスリストの作成にかかります．しばらくすると，実行例のようなリストが得られます．左から，そのファイル名で指定したプログラムで使われているラベル名，そのラベル名の付いている行番号，そのラベル名を参照している行番号の順で表示されます．当然，ラベルを一つも使っていないプログラムでは，ラベル名の表示はされません．

それでは，このプログラムを存分に活用して，あなたのプログラム開発に御利用ください．

```
1000 '
1010 ' label cross reference
1020 '
1030 ON ERROR GOTO *ERRTRAP0
1040 DEFINT A-Z
1050 DEF FNSE(X,X$)=INSTR(X,IN$,X$)
1060 DEF FNSESP(X)=FNSE(X," ")
1070 DEF FNSEQU(X)=FNSE(X,CHR$(34))
1080 '
1090 N=8 : M=50
1100 DIM KWORD$(N),KLEN(N),WORD$(M),WLINE(M),WCELL(M),CELL(1,M*5)
1110 FOR I=0 TO N
1120  READ I$ : KWORD$(I)=I$ : KLEN(I)=LEN(I$)
1130 NEXT
1140 WORD=-1
1150 PRINT " **** label cross reference for basic ****"
1160 PRINT " input basic text must be ascii saved file"
1170 '
1180 *RECOVERY
1190  PRINT
1200  INPUT "enter file name ";FLNAME$
1210  GOSUB *PASS1
1220  GOSUB *PASS2
1230  GOSUB *DISP
1240 END
1250 '
1260 *SKIP
1270  WHILE MID$(IN$,I,1)=" "
1280   I=I+1
1290  WEND
1300  RETURN
1310 '
1320 *GETLABEL
1330  I=I+1 : K=I : K$=MID$(IN$,K,1)
1340  WHILE "A"=<K$ AND K$=<"Z" OR "0"=<K$ AND K$=<"9" OR K$="."
1350   K=K+1 : K$=MID$(IN$,K,1)
1360  WEND
1370  LABEL$=MID$(IN$,I,K-I)
1380  RETURN
1390 '
1400 *SETLN
```

```
1410   GOSUB *SKIP : K=I
1420   IF MID$(IN$,I,1)<>"*" THEN RETURN
1430   GOSUB *GETLABEL
1440   I=0 : GOOD=-1
1450   WHILE GOOD AND I=<WORD
1460    IF WORD$(I)=LABEL$ THEN GOOD=0 ELSE I=I+1
1470   WEND
1480   IF GOOD THEN PRINT "undefined label *";LABEL$;" in ";LN : RETURN
1490   C=WCELL(I) : WCELL(I)=CELL
1500   CELL(0,CELL)=C : CELL(1,CELL)=LN
1510   CELL=CELL+1 : IF CELL>M*5 THEN *ABNORMAL
1520   RETURN
1530 '
1540 *ABNORMAL
1550   PRINT
1560   PRINT "too many labels !! , then can't go on"
1570   END
1580 '
1590 *PASS1
1600 OPEN FLNAME$ FOR INPUT AS #1
1610  PRINT "pass-1 ";
1620  WHILE NOT EOF(1)
1630   LINE INPUT#1,IN$
1640   I=FNSESP(1) : LN=VAL(LEFT$(IN$,I-1))
1650   GOSUB *SKIP
1660   IF MID$(IN$,I,1)<>"*" THEN GOTO *PASS11
1670    GOSUB *GETLABEL
1680    WORD=WORD+1
1690    WORD$(WORD)=LABEL$ : WLINE(WORD)=LN
1700    WCELL(WORD)=-1
1710    IF WORD>=M THEN *ABNORMAL
1720   *PASS11
1730  WEND
1740  CLOSE
1750  PRINT "end"
1760  RETURN
1770 '
1780 *PASS2
1790  PRINT "pass-2 ";
1800  OPEN FLNAME$ FOR INPUT AS #1
1810  WHILE NOT EOF(1)
1820   LINE INPUT#1,IN$ : IN$=IN$+" " : I=FNSESP(1)
1830   LN=VAL(LEFT$(IN$,I-1)) : I=FNSEQU(1)
1840   WHILE I<>0
1850    J=FNSEQU(I+1) : IF J=0 THEN J=LEN(IN$)
1860    IN$=LEFT$(IN$,I-1)+MID$(IN$,J+1,255)
1870    I=FNSEQU(1)
1880   WEND
1890   I=FNSE(1,"REM ") : IF I<>0 THEN IN$=LEFT$(IN$,I-1)
1900   I=FNSE(1,"'") : IF I<>0 THEN IN$=LEFT$(IN$,I-1)
1910   FOR KW=0 TO N
1920    J=0
1930  *RETRY : I=FNSE(J+1,KWORD$(KW))
1940    IF I=0 THEN *NEXTWORD
1950    J=I : I=I+KLEN(KW) : GOSUB *SETLN
1960  *RETRY1
1970    I=K : GOSUB *SKIP : IF MID$(IN$,I,1)<>"," THEN *RETRY
1980    I=I+1 : GOSUB *SETLN : GOTO *RETRY1
1990  *NEXTWORD
2000   NEXT
2010  WEND
2020  CLOSE
2030  PRINT "end"
2040  RETURN
2050 '
2060 *DISP
2070  PRINT
2080  PRINT "     label name       defined   referenced lines"
2090  PRINT
2100  FOR I=0 TO WORD
2110   PRINT WORD$(I);TAB(20);
2120   PRINT USING" ; ##### ; ";WLINE(I);
2130   J=WCELL(I)
2140   WHILE J>=0
2150    PRINT USING"##### ";CELL(1,J);
```

```
2160      J=CELL(0,J)
2170     WEND
2180     PRINT
2190    NEXT
2200    RETURN
2210 '
2220 DATA GOSUB,RETURN,GOTO,GO TO,THEN,ELSE,RESUME,RESTORE,RUN
2230 '
2240 *ERRTRAP0
2250  IF ERL<>1600 OR (ERR<>56 AND ERR<>53) GOTO *ERRTRAP1
2260  PRINT:PRINT "File not Found.":BEEP 1:BEEP 0:RESUME *RECOVERY
2270 '
2280 *ERRTRAP1
2290  IF ERL<>1600 OR ERR<>62 GOTO *ERRTRAP2
2300  DR=VAL(MID$(FLNAME$,1,1)):IF DR=0 THEN DR=1
2310  PRINT:PRINT USING "Disk offline. [ drive # ]";DR:BEEP 1:BEEP 0
2320  INPUT WAIT 50,"Retry";IN$:RESUME *RECOVERY
2330 '
2340 *ERRTRAP2
2350  ON ERROR GOTO 0
```

```
run
 **** label cross reference for basic ****
 input basic text must be ascii saved file

enter file name ? cross.asc
pass-1 end
pass-2 end
      label name          defined     referenced lines

RECOVERY                ;   1180  ;  2320   2260
SKIP                    ;   1260  ;  1970   1650   1410
GETLABEL                ;   1320  ;  1670   1430
SETLN                   ;   1400  ;  1980   1950
ABNORMAL                ;   1540  ;  1710   1510
PASS1                   ;   1590  ;  1210
PASS11                  ;   1720  ;  1660
PASS2                   ;   1780  ;  1220
RETRY                   ;   1930  ;  1970
RETRY1                  ;   1960  ;  1980
NEXTWORD                ;   1990  ;  1940
DISP                    ;   2060  ;  1230
ERRTRAP0                ;   2240  ;  1030
ERRTRAP1                ;   2280  ;  2250
ERRTRAP2                ;   2340  ;  2290
OK
■
```

APPENDIX

```
1000 '
1010 '  graphic color box demo_program by cat
1020 '
1030  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
1040 CLEAR ,&HE500
1050 DEFINT A-Z
1060 DIM C(4),TC(4,4),P(997),P1(510),P2(510),P3(510),P4(412),P5(412)
                        ,M1(510),M2(510),M3(510),M4(412),M5(412)
1070 SCREEN 0,3 : CLS 3 : SCREEN ,0 : CONSOLE 0,25,0,1
1080 ON ERROR GOTO *ERRTRAP
1090 '
1100  GOSUB *TILEINIT
1110  GOSUB *MAKEPAT
1120  RESTORE *BOXDATA
1130 '
1140 *LOOP
1150  READ BOXTYPE,X,Y : X=X*48 : Y=Y*12
1160  IF BOXTYPE THEN *BOX1
1170   GOSUB *COLORUP
1180   PUT@(1,1),M3,AND : PUT@(X,Y),P3,AND
1190   GET@(1,1)-(101,26),P : PUT@(X,Y),P,OR
1200   GOTO *BOX2
1210  '
1220  *BOX1
1230   GOSUB *COLORUP
1240   PUT@(1,1),M1,AND : PUT@(X,Y),P1,AND
1250   GET@(1,1)-(101,26),P : PUT@(X,Y),P,OR
1260   GOSUB *COLORUP
1270   PUT@(1,1),M2,AND : PUT@(X,Y),P2,AND
1280   GET@(1,1)-(101,26),P : PUT@(X,Y),P,OR
1290  '
1300  *BOX2
1310   GOSUB *COLORUP
1320   PUT@(1,13),M4,AND : PUT@(X,Y+12),P4,AND
1330   GET@(1,13)-(51,51),P : PUT@(X,Y+12),P,OR
1340   GOSUB *COLORUP
1350   PUT@(51,13),M5,AND : PUT@(X+50,Y+12),P5,AND
1360   GET@(51,13)-(101,51),P : PUT@(X+50,Y+12),P,OR
1370  '
1380  GOTO *LOOP
1390 '
1400 *COLORUP
1410   LINE (0,0)-(102,52),0,BF
1420   READ RED,GREEN,BLUE
1430   GOSUB *MAKETILE
1440   LINE (0,0)-(102,52),7,B : PAINT(50,25),TILE$,7
1450  RETURN
1460 '
1470 '  -- matrix pattern reader --
1480 '
1490 *TILEINIT
1500   RESTORE *TILEDATA
1510   FOR SIZE=1 TO 2
1520     FOR BIT=1 TO 4
1530       READ SFT(SIZE,BIT)
1540     NEXT
1550   NEXT
1560   *TILEDATA : DATA 1,3,2,4,2,4,1,3
1570  RETURN
1580 '
1590 '   -- color bit breaker --
```

```
1600 '
1610 *MAKETILE
1620  C(0)=BLUE : C(1)=RED : C(2)=GREEN
1630   FOR BANK=0 TO 2
1640     FOR BIT=1 TO 4
1650       TC(BANK,BIT)=0
1660       IF C(BANK)<1 THEN 1680
1670       TC(BANK,BIT)=1:C(BANK)=C(BANK)-1
1680     NEXT
1690   NEXT
1700 '
1710 '  -- tile genarater --
1720 '
1730   TILE$=""
1740   FOR SIZE=1 TO 2
1750     FOR BANK=0 TO 2
1760       TILE(SIZE,BANK)=0
1770       FOR BIT=1 TO 4
1780         TILE(SIZE,BANK)=TILE(SIZE,BANK)+TC(BANK,SFT(SIZE,BIT))*2^(BIT-1)
1790       NEXT
1800       TILE(SIZE,BANK)=TILE(SIZE,BANK)*16+TILE(SIZE,BANK)
1810       TILE$=TILE$+CHR$(TILE(SIZE,BANK))
1820     NEXT
1830   NEXT
1840  RETURN
1850 '
1860 ' -- make get put window pattern --
1870 '
1880 *MAKEPAT
1890  LINE(0,0)-(100,50),7,BF:GET@(0,0)-(100,50),P
1900 '
1910  LINE(50,0)-(50,24),0:LINE-(2,12),0:LINE-(50,0),0
1920  PAINT(40,10),0,0:LINE (50,0)-(50,24),7:GET@(0,0)-(100,25),P1
1930  PUT@(0,0),P,XOR:GET@(0,0)-(100,25),M1
1940 '
1950  LINE(0,0)-(100,50),7,BF:LINE(50,0)-(98,12),0:LINE-(50,24),0
1960  LINE-(50,0),0:PAINT (60,10),0,0:GET@ (0,0)-(100,25),P2
1970  PUT@(0,0),P,XOR:GET@(0,0)-(100,25),M2
1980 '
1990  LINE(0,0)-(100,50),7,BF:LINE(50,0)-(98,12),0:LINE -(50,24),0
2000  LINE-(2,12),0:LINE-(50,0),0:PAINT(50,10),0,0:GET@(0,0)-(100,25),P3
2010  PUT@(0,0),P,XOR:GET@(0,0)-(100,25),M3
2020 '
2030  LINE(0,0)-(100,50),0,BF:LINE(0,0)-(100,50),7,B:LINE(2,12)-(50,24),7
2040  LINE-(50,49),7:LINE-(2,37),7:LINE-(1,37),7:LINE-(1,12),7
2050  LINE-(2,12),7:PAINT(60,5),7,7:PAINT(1,49),7,7:GET@(0,12)-(50,50),P4
2060  PUT@(0,0),P,XOR:GET@(0,12)-(50,50),M4
2070 '
2080  LINE(0,0)-(100,50),0,BF:LINE(0,0)-(100,50),7,B:LINE(50,24)-(98,12),7
2090  LINE-(99,12),7:LINE-(99,37),7:LINE-(98,37),7:LINE-(50,49),7:LINE-(49,49),7
2100  LINE-(49,24),7:LINE-(50,24),7:PAINT(40,5),7,7:PAINT(99,49),7,7
2110  GET@(50,12)-(100,50),P5:PUT@(0,0),P,XOR:GET@(50,12)-(100,50),M5
2120 '
2130 RETURN
2140 '
2150 *ERRTRAP
2160  IF ERL<>1150 THEN ON ERROR GOTO 0
2170  LINE (0,0)-(102,52),0,BF
2180  END
2190
2200 *BOXDATA
2210  DATA  0, 4,6,  3,3,1,  1,1,1,  1,1,1
2220  DATA  0, 3,7,  1,1,1,  1,1,1,  1,4,2
2230  DATA  0, 4,0,  1,1,4,  1,1,1,  0,1,4
2240  DATA  0, 3,5,  1,1,1,  2,1,2,  2,1,1
2250  DATA  0, 3,3,  1,1,1,  1,2,2,  0,2,1
2260  DATA  1, 3,1,  1,1,2,  1,2,1,  1,3,4,  1,1,3
2270  DATA  0, 2,8,  3,2,1,  1,1,1,  0,4,2
2280  DATA  0, 1,11,1,1,1,  1,1,4,  1,1,1
2290  DATA  0, 1,9,  2,1,0,  1,2,4,  0,4,1
2300  DATA  1, 2,12,1,4,2,  1,3,2,  0,1,4,  0,3,3
2310  DATA  0, 5,7,  2,3,0,  1,2,1,  1,1,1
2320  DATA  0,10,4,  4,2,2,  1,1,1,  4,3,3
2330  DATA  0, 9,5,  3,1,2,  1,1,1,  4,1,2
2340  DATA  0, 8,6,  3,0,2,  1,1,1,  4,0,2
2350  DATA  0, 7,7,  3,0,0,  1,1,1,  4,0,0
2360  DATA  0, 6,8,  1,1,1,  1,3,1,  1,1,1
2370  DATA  0, 7,11,1,1,1,  0,3,0,  1,1,1
2380  DATA  0, 8,12,3,0,3,  0,4,0,  4,0,4
2390  DATA  0, 7,9,  3,3,1,  1,4,1,  4,1,4
2400  DATA  0, 6,6,  1,1,1,  2,2,1,  0,3,3
2410  DATA  0, 6,4,  4,4,0,  4,4,1,  0,4,4
```

```
1000 '
1010 ' graphic color box demo_program by cat
1020 '
1030 FOR I=0 TO 7 : COLOR=(I,I) : NEXT
1040 CLEAR ,&HE500
1050 DEFINT A-Z
1060 DIM TILE(2,2),C(4),TC(4,4),P(1036),P1(347),P2(347),P3(347),P4(347),P5(347)
                                   ,M1(347),M2(347),M3(347),M4(347),M5(347)
1070 SCREEN 0,3 : CLS 3 : SCREEN ,0 : CONSOLE 0,25,0,1 : PAI=3
1080 ON ERROR GOTO *ERRTRAP
1090 '
1100  GOSUB *TILEINIT
1110  GOSUB *MAKEPAT
1120  SCREEN 0,0
1130  RESTORE *BOXDATA
1140 '
1150 *LOOP
1160  READ ENDCODE,X,Y : X=X*60 : Y=Y*15
1170  IF ENDCODE THEN LINE (0,0)-(102,53),0,BF : END
1180   GOSUB *COLORUP
1190   PUT@(1,1),M1,AND,7,0 : PUT@(X,Y),P1,AND,7,0
1200   GET@(1,1)-(101,52),P : PUT@(X,Y),P,OR
1210  '
1220   GOSUB *COLORUP
1230   PUT@(1,1),M2,AND,7,0 : PUT@(X,Y),P2,AND,7,0
1240   GET@(1,1)-(101,52),P : PUT@(X,Y),P,OR
1250  '
1260   GOSUB *COLORUP
1270   PUT@(1,1),M3,AND,7,0 : PUT@(X,Y),P3,AND,7,0
1280   GET@(1,1)-(101,52),P : PUT@(X,Y),P,OR
1290  '
1300   GOSUB *COLORUP
1310   PUT@(1,1),M4,AND,7,0 : PUT@(X,Y),P4,AND,7,0
1320   GET@(1,1)-(101,52),P : PUT@(X,Y),P,OR
1330  '
1340   GOSUB *COLORUP
1350   PUT@(1,1),M5,AND,7,0 : PUT@(X,Y),P5,AND,7,0
1360   GET@(1,1)-(101,52),P : PUT@(X,Y),P,OR
1370  '
1380  GOTO *LOOP
1390 '
1400 *COLORUP
1410   LINE (0,0)-(102,53),0,BF
1420   READ RED,GREEN,BLUE
1430   GOSUB *MAKETILE
1440   LINE (0,0)-(102,53),7,B : PAINT(50,25),TILE$,7
1450  RETURN
1460 '
1470 ' -- matrix pattern reader --
1480 '
1490 *TILEINIT
1500   RESTORE *TILEDATA
1510   FOR SIZE=1 TO 2
1520     FOR BIT=1 TO 4
1530       READ SFT(SIZE,BIT)
1540     NEXT
1550   NEXT
1560   *TILEDATA : DATA 1,3,2,4,2,4,1,3
1570  RETURN
1580 '
1590 '  -- color bit breaker --
1600 '
1610 *MAKETILE
1620  C(0)=BLUE : C(1)=RED : C(2)=GREEN
1630   FOR BANK=0 TO 2
1640     FOR BIT=1 TO 4
1650       TC(BANK,BIT)=0
1660       IF C(BANK)<1 THEN 1680
1670       TC(BANK,BIT)=1:C(BANK)=C(BANK)-1
1680     NEXT
1690   NEXT
1700 '
1710 '  -- tile genarater --
1720 '
1730   TILE$=""
1740   FOR SIZE=1 TO 2
1750     FOR BANK=0 TO 2
1760       TILE(SIZE,BANK)=0
1770       FOR BIT=1 TO 4
1780         TILE(SIZE,BANK)=TILE(SIZE,BANK)+TC(BANK,SFT(SIZE,BIT))*2^(BIT-1)
1790       NEXT
1800       TILE(SIZE,BANK)=TILE(SIZE,BANK)*16+TILE(SIZE,BANK)
1810       TILE$=TILE$+CHR$(TILE(SIZE,BANK))
1820     NEXT
1830   NEXT
1840  RETURN
```

```
1850 '
1860 ' -- make get put window pattern --
1870 '
1880 *MAKEPAT
1890  SCREEN 1
1900  LINE(50,25)-(150,76),7,BF : GET@(50,25)-(150,76),P
1910  CIRCLE(100,50),49,0,PAI*13/16,PAI*2
1920  CIRCLE(105,45),49,0,PAI*7/8,PAI*30/16
1930  PAINT(90,70),0,0 : GET@(50,25)-(150,76),P1
1940  PUT@(50,25),P : GET@(50,25)-(150,76),M1
1950 '
1960  LINE(50,25)-(150,76),7,BF
1970  CIRCLE(100,50),49,0
1980  CIRCLE(105,45),50,0,PAI*15/16,PAI*15/8
1990  CIRCLE(107,45),40,0,PAI*5/8,0
2000  PAINT(90,65),0,0
2010  CIRCLE(100,50),49,7 : GET@(50,25)-(150,76),P2
2020  PUT@(50,25),P : GET@(50,25)-(150,76),M2
2030 '
2040  LINE(50,25)-(150,76),7,BF
2050  CIRCLE(100,50),49,0
2060  CIRCLE(107,45),41,0,PAI*5/8,0
2070  CIRCLE(111,43),30,0
2080  PAINT(90,60),0,0
2090  CIRCLE(100,50),49,7 : GET@(50,25)-(150,76),P3
2100  PUT@(50,25),P : GET@(50,25)-(150,76),M3
2110 '
2120  LINE(50,25)-(150,76),7,BF
2130  CIRCLE(111,43),31,0
2140  CIRCLE(114,42),20,0
2150  PAINT(90,50),0,0 : GET@(50,25)-(150,76),P4
2160  PUT@(50,25),P : GET@(50,25)-(150,76),M4
2170 '
2180  LINE(50,25)-(150,76),7,BF
2190  CIRCLE(114,42),21,0
2200  PAINT(114,42),0,0 : GET@(50,25)-(150,76),P5
2210  PUT@(50,25),P : GET@(50,25)-(150,76),M5
2220  LINE(50,25)-(150,76),0,BF
2230 '
2240 RETURN
2250 '
2260 *ERRTRAP
2270  IF ERL<>1160 THEN ON ERROR GOTO 0
2280  LINE (0,0)-(102,52),0,BF
2290  END
2300 '
2310 *BOXDATA :'     Blue Red Green  .........
2320  DATA  0,1,5, 0,0,1, 0,0,2, 0,0,3, 0,0,4, 2,2,4
2330  DATA  0,1,8, 0,1,0, 0,2,0, 0,3,0, 0,4,0, 2,4,2
2340  DATA  0,2,9, 1,0,0, 2,0,0, 3,0,0, 4,0,0, 4,2,2
2350  DATA  0,2,4, 0,1,1, 0,2,2, 0,3,3, 0,4,4, 2,4,4
2360  DATA  0,3,5, 1,1,0, 2,2,0, 3,3,0, 4,4,0, 4,4,2
2370  DATA  0,3,8, 1,0,1, 2,0,2, 3,0,3, 4,0,4, 4,2,4
2380  DATA  0,4,4, 1,1,2, 1,1,3, 1,1,4, 2,2,4, 3,3,4
2390  DATA  0,4,9, 1,2,1, 1,3,1, 1,4,1, 2,4,2, 3,4,3
2400  DATA  0,5,5, 2,1,1, 3,1,1, 4,1,1, 4,2,2, 4,3,3
2410  DATA  0,5,8, 1,1,0, 2,2,0, 3,3,0, 4,4,0, 4,4,2
2420  DATA  0,8,7, 1,1,2, 1,1,3, 1,1,4, 2,2,4, 3,3,4
2430  DATA  1,0,0
```

索　　引

サ

タ

ナ

ハ

マ

PC-8801 BASIC入門
アスキー・システム・バンク(PC88＃1)

1982年12月20日　第1版第7刷発行
定価2,500円

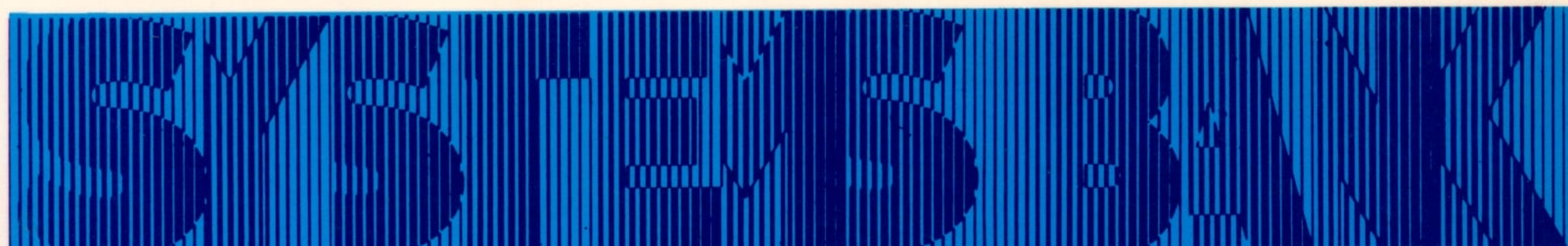
著　者　工藤丈彦・屋敷誠二・横溝和宏
発行者　塚本慶一郎
発行所　株式会社アスキー
〒107　港区南青山5－11－5住友青山ビル5F
振　替　東京7-57496
電　話　03－486－7111(代表)

©1982 ASCII Corporation. Printed in Japan.

本書は著作権法上の保護を受けています．本書の一部あるいは全部について(ソフトウェア及びプログラムを含む)，株式会社アスキーから文書による許諾を得ずに，いかなる方法においても無断で複写，複製することは禁じられています．

編集担当者　井芹昌信
印刷　三共グラビヤ印刷

ISBN4-87148-290-1 C3055 ¥2500E

ASCII SYSTEM BANK

PC88編第①巻 内容抜粋

PC-8801を電卓として使う

間接モード(プログラムモード)で使う

プログラムを編集してみよう

N88-BASICで扱えるデータ

変数の名前(変数名)と型

EDITコマンドを使う

便利なコントロール機能

グラフィックス・ワールド

2つの座標

LINE,STEP,POINT —直線を描く—

もう1つのCOLOR(パレット)—色を自由に操る—

WINDOW(ワールド座標とは)—座標を自由に設定する—

VIEW(スクリーン座標とは)—表示画面の大きさを自由に設定する—

POINT関数,MAP関数—便利な関数その1—

GET@, PUT@—グラフィックパターン,漢字を操る—

WINDOW関数,VIEW関数—便利な関数その2—

N88-BASICで扱える入出力装置

割り込みって何でしょう

BASICで割り込みを使うには

プログラマブル・ファンクションキーを割り込みで使う

HELPキーを割り込みで使う

STOPキーを割り込みで使う

タイマを割り込みで使う

ライトペンを割り込みで使う

使い易いプログラムを作るために—入力編—

使い易いプログラムを作るために—出力編—

デバッギングのノウハウ

ラベルクロスリファレンス

工藤丈彦・屋敷誠二
横溝和宏　　共著

アスキー・システム・バンク(PC88#1)

PC-8801 BASIC入門

定価2,500円

ISBN4-87148-290-1

C3055 ¥2500E